Multi-Party Private Set Operations from Predicative Zero-Sharing

Minglang Dong[1,2], Yu Chen[1,2,3], Cong Zhang[4], Yujie Bai[1,5], and Yang Cao[1,2]

[1] School of Cyber Science and Technology, Shandong University, Qingdao, China
[2] State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, Qingdao, China
[3] State Key Laboratory of Cryptology, Beijing, China
[4] Institute for Advanced Study, Tsinghua University, Beijing, China
[5] Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security, Hangzhou, China
{minglang_dong,caoyang24}@mail.sdu.edu.cn, yuchen.prc@gmail.com,
zhangcong@mail.tsinghua.edu.cn, baiyujie@bcds.org.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. With computation and communication complexity scaling linearly with the set size n, 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), for the first time in the standard semi-honest model. Furthermore, the instantiations of our framework, which primarily rely on symmetric-key techniques, provide efficient protocols for MPSI, MPSI-card, MPSI-card-sum, and multi-party private set union (MPSU), with online performance that either surpasses or matches the state of the art in standard semi-honest model.

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, 57, 39, 56, 5, 32, 38, 34, 29, 28, 41, 11, 3, 60, 1, 64, 61, 27], which is to compute intersection, and its variants — MPSI with cardinality output (MPSI-card) [36, 5, 15, 26, 30, 27], which is to compute intersection cardinality, MPSI with cardinality and sum (MPSI-card-sum) [15, 30], which is to compute intersection cardinality and sum (of the associated payloads), and circuit-MPSI [36, 5, 11, 64, 59], 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, 5, 58, 60, 40, 25, 17], which is to compute union, and its variants — MPSU with cardinality output (MPSU-card) [5], which is to compute union cardinality, and circuit-MPSU [5], 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 practical (even state-of-the-art) solutions [40, 64, 26, 27, 59] rely on non-collusion assumptions 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.
– **Unmet application demands.** Many real-world applications require more than just the computation of intersection and union (or partial/aggregate information derived from them). However, existing protocols are limited in either functionality or efficiency to meet these broader needs. For instance, a public health agency wants to identify eligible individuals for a new health program based on both their medical conditions and income levels. Hospitals maintain private records of patients with certain conditions, while welfare departments manage income-level data. The task requires computing the union of patients with a specific condition across different hospitals, while ensuring privacy, and then performing an intersection with the income eligibility records [36]. This type of composite set operation — computing a union followed by an intersection— goes beyond the above two categories, which focus solely on either intersection or union. To date, practical solutions for such composite operations remain absent in the MPSO literature.
– **Fragmented landscape of protocols.** Most existing works focus on only one or two specific functionalities, leading to a proliferation of isolated schemes and the absence of a unified framework for 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 $X_2 \setminus (X_1 \cap X_3)$ 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 [5] redesigned the circuits for computing intersection, union and difference as oblivious sorting and adjacent comparisons, with implementations using generic MPC protocols. The generic MPC technique allows their protocols to be composable to compute arbitrary compositions of binary set operations, and thereby fully realizing MPSO functionality. However, this generality comes at the cost of substantial computational overhead. Even the simplest cases, such as MPSI, demonstrate poor practical efficiency — for instance, the largest-scale reported experiment for MPSI involves only 3 parties, each holding 2^{11} items of 32 bits, however, takes 24.8 seconds to complete. 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 good efficiency in the semi-honest model?

1.1 Our Contributions

In this work, we answer the above question affirmatively. Our technical route proceeds as follows: First, we define a predicate formula representation for any set formulas; Second, we introduce a composable primitive — predicative zero-sharing — and its composition technique; Then, we instantiate predicative zero-sharing as a primitive tailored for MPSO — membership zero-sharing — using lightweight building blocks; Finally,

with membership zero-sharing serving as the main building block, we present 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 [5] 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 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 simplified 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 simplified security definition simplifies the security proof of our predicative zero-sharing protocols. Moreover, under this simplified security definition, we can relax predicative zero-sharing's security 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], 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 with standard security 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_{pivot}) inputs an element and the other parties input sets. The output secret-sharing among the parties encodes whether P_{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 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. We construct these two relaxed membership zero-sharing instantiations using lightweight components, including oblivious programmable pseudorandom function (OPPRF) [38, 45, 12, 54, 48], batch secret-shared private membership test (batch ssPMT) [17], and ROT. Membership zero-sharing bridges the gap between generality and practicality for MPSO. Its composability, inherited from predicative zero-sharing, facilitates our framework

to compute arbitrary set operations, while its efficient instantiations contribute to the good efficiency. The technical route is outlined in Figure 1.

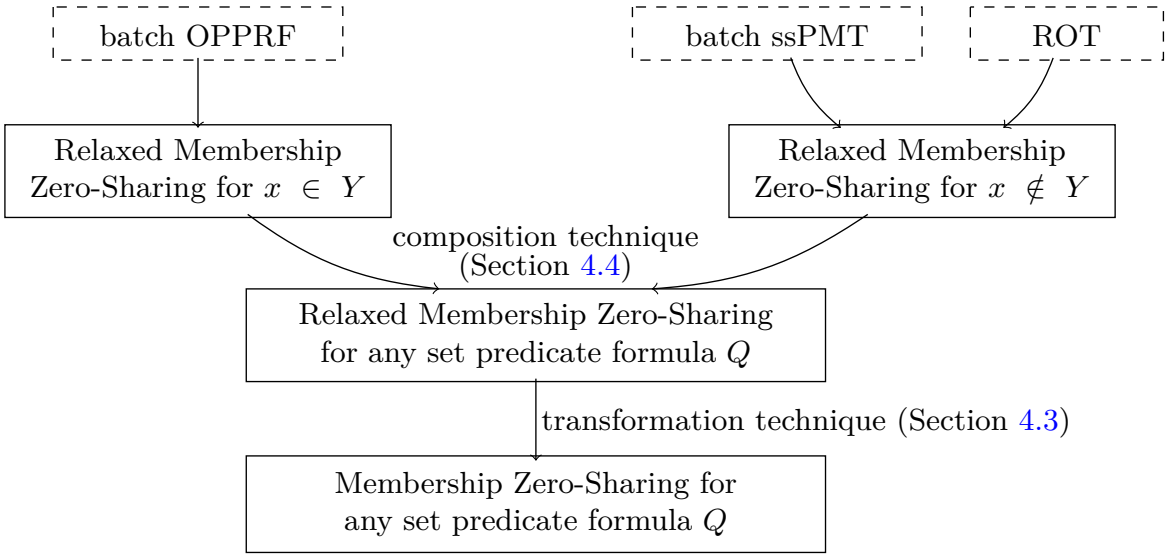

Fig. 1. Technical route of building membership zero-sharing protocols for any first-order set predicate formulas. The newly introduced primitives are marked with solid boxes. The existing primitives are marked with dashed boxes.

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. At a high level, our framework can be abstracted as two phases: The first phase involves several secret-sharing processes, where each process enables the parties to somehow obtain secret-sharings corresponding to one of the disjoint sets Y_i from a partition $\{Y_1, \cdots, Y_s\}$ of the desired set Y. The partition is determined by the CPF representation of Y, where Y_i is the set represented by each subformula Q_i. The structural properties of our CPF representation also guarantee that each disjoint set Y_i from the partition is a subset of some party P_j's input set X_j (see Section 3). Therefore, the secret-sharing process for Y_i is realized by the parties obliviously assisting P_j to test the membership of each element in X_j with respect to Y and encode the test result into a secret-sharing. The second phase is to reconstruct all secrets from secret-sharing processes to the leader, who combines them to reveal Y. To elaborate on our framework, we begin with the simplest case, where Y is a subset of the leader's input set.[6] The simplicity of this case lies in the fact that the leader already has all elements in Y, thus, to compute Y, it suffices for the leader to obtain a vector indicating which elements in its set belong Y, using a designated value (e.g., 0). Since the CPF representation yields a trivial partition with Y itself, the parties only need to assist the leader to encode the membership test result of each of its elements with respect to Y. Specifically, the leader acts as P_{pivot}, and for each element x in its input set, the parties invoke the membership zero-sharing associated with the membership testing set predicate formula derived from the CPF representation. As a result, if $x \in Y$, the parties hold a secret-sharing of 0; otherwise, the parties hold a random secret-sharing. This construction can be optimized using the hashing-to-bins technique (see

[6] MPSI is a typical example of this case, as the intersection is a subset of any input sets.

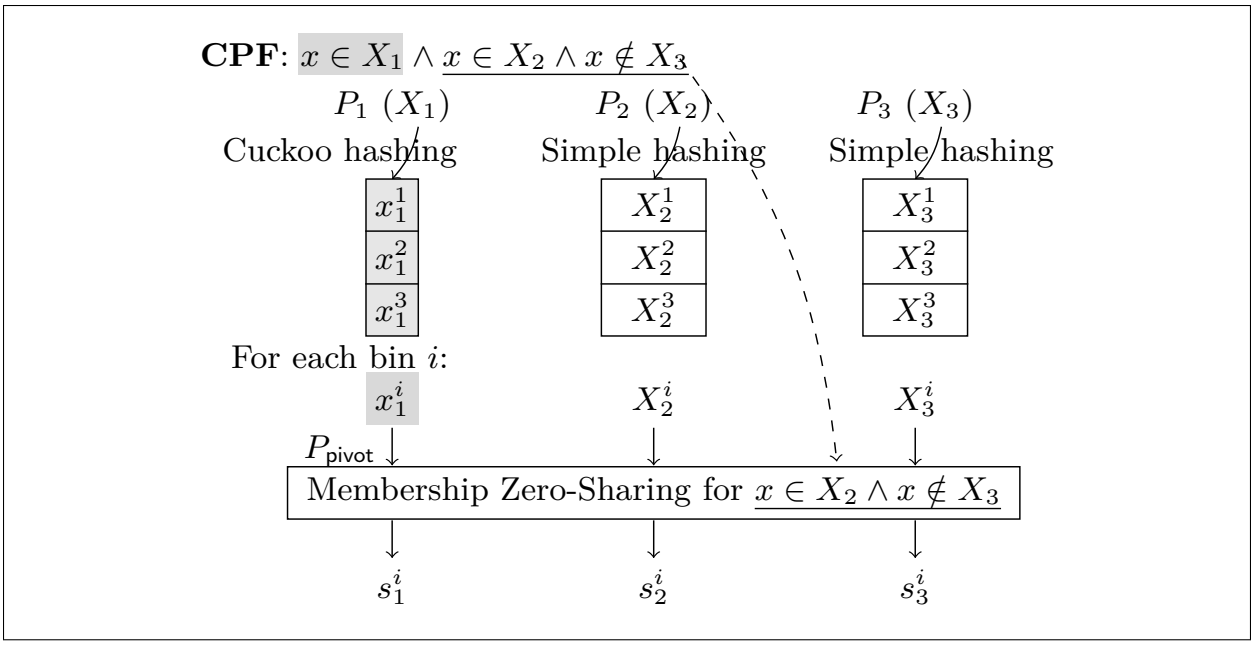

Fig. 2. An example of the simplest case of our framework, where the set formulas being computed is $(X_1 \cap X_2) \setminus X_3$. If the element x_1^i in X_1 satisfies $x_1^i \in X_2^i \wedge x_1^i \notin X_3^i$ (i.e. $x_1^i \in (X_1 \cap X_2) \setminus X_3$), $s_1^i + s_2^i + s_3^i = 0$.

Figure 2). In the second phase, the parties reconstruct all secret-sharings to the leader, who computes Y by identifying all elements whose corresponding secret-sharings reconstructed to 0. We proceed to generalize this simplest case to achieve MPSO functionality. For each subformula Q_i, the parties whose sets are involved in Q_i, engage in the secret-sharing process for Y_i as follows: P_j acts as P_{pivot}, and for each element x in X_j, the parties invoke the membership zero-sharing associated with the membership testing set predicate formula derived from Q_i. The invocation is similar to the simplest case (see Figure 3). P_j adds $x \| 0^{l'}$ to its share. For the parties whose sets are not involved in Q_i, they set their corresponding shares to 0. In the end, for each element x in X_j, if $x \in Y_i$, all parties hold a secret-sharing of $x \| 0^{l'}$;[7] otherwise, they hold a random secret-sharing. In the general case, a straightforward reconstruction for the second phase may leak information to the leader about which party each reconstructed element originates from, through the order of secret-sharings. To address the leakage, the parties invoke multi-party secret-shared shuffle to randomly permute and re-share all secret-sharings. Finally, the shuffled secret-sharings are reconstructed to the leader. Since Y remains being secret-shared before the last reconstruction step, this MPSO protocol is easy to be extended to MPSO-card and circuit-MPSO protocols.

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 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 the standard semi-honest model. The previous

[7] The appended $0^{l'}$ is an all-zero string of length l' to help the leader distinguish the valid element x and a random secret during the reconstruction

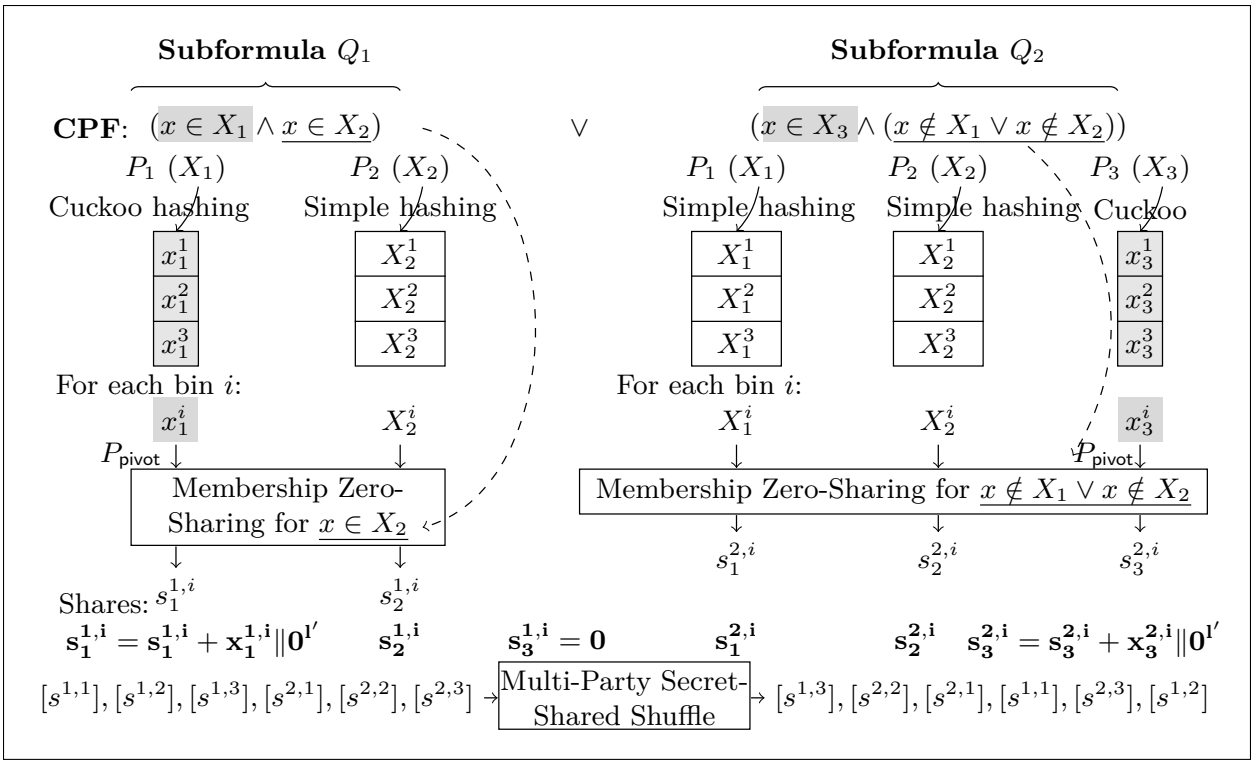

Fig. 3. An example of the general case of our framework, where the set formulas being computed is $(X_1 \cap X_2) \cup X_3$. The set $X_1 \cap X_2$ represented by Q_1 and the set $X_3 \setminus (X_1 \cap X_2)$ represented by Q_2 form a partition of $(X_1 \cap X_2) \cup X_3$. If the element x_1^i in X_1 satisfies $x_1^i \in X_2^i$ (i.e. $x_1^i \in X_1 \cap X_2$), $s_1^{1,i} + s_2^{1,i} + s_3^{1,i} = x_1^i \| 0^{l'}$; If the element x_3^i in X_3 satisfies $x_3^i \notin X_1^i \vee x_3^i \notin X_2^i$ (i.e. $x_3^i \in X_3 \setminus (X_1 \cap X_2)$), $s_1^{2,i} + s_2^{2,i} + s_3^{2,i} = x_3^i \| 0^{l'}$.

MPSI protocol [38] with this optimal complexity is only secure in the augmented semi-honest model, which provides weaker security than the standard semi-honest model (For an introduction to the augmented semi-honest model and its relation to the semi-honest model, see Section 2.4.4 of [31]). In this work, we close this gap. Our MPSI protocol is also the most online-efficient MPSI protocol to date, which is $2.4 - 5.2\times$ (resp. $1.1 - 2.6\times$) faster than the state-of-the-art MPSI protocol [61] in LAN (resp. WAN) setting. Concretely, it requires only 8.9 seconds in online phase for 10 parties with sets of 2^{20} items each, regardless of the bit length of items.

MPSI-card, MPSI-card-sum and Circuit-MPSI. The MPSI-card and MPSI-card-sum protocols from our framework are the first constructions with the optimal computation and communication complexity in the standard semi-honest model. Our MPSI-card is the most online-efficient MPSI-card protocol to date, with $14.0 - 20.3\times$ lower communication than the state of the art [15]. Our MPSI-card-sum provides the first MPSI-card-sum implementation and 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. The circuit-MPSI protocol from our framework is the first circuit-MPSI construction in dishonest majority setting. Notably, while circuit-MPSI has been considered before for honest majority setting, this is the first work to consider the problem 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, 17]. 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 has the lowest online communication costs to date, with an improvement up to 1.8× compared with the state-of-the-art MPSU protocol [17]. In bandwidth-constrained networks, its online performance becomes increasingly competitive as the number of parties grows.

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.

Differences between the Conference and Full Versions. In addition to including the details omitted from the conference version (e.g., formal proofs of all theorems), an important revision in this full version is that we introduce another MPSO framework based on public-key operations in the standard semi-honest model (Appendix H). This framework builds upon a public-key encryption (PKE) based counterpart of membership zero-sharing — called predicative zero-encryption — as its main building block and fully realizes the MPSO and MPSO-card functionalities. Although the reliance on public-key operations renders this framework less efficient in practice than the one in the main body, it achieves superior asymptotic complexity.

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, 5, 58, 11, 40, 64, 26, 59], or have non-negligible false positives [1, 60]. We only focus on works achieving semi-honest security against arbitrary collusion without incurring non-negligible false positives. Distribution of research attention among these works is extremely imbalanced: MPSI has been extensively studied [22, 36, 57, 56, 32, 38, 34, 29, 28, 41, 3, 61], while MPSU only receives relatively little attention [36, 23, 25, 17]. MPSI-card and MPSI-card-sum are extremely understudied sub-fields, with only a couple of secure MPSI-card [36, 15, 30] and MPSI-card-sum [15, 30] protocols 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 F.

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

Kolesnikov et al. [38] proposed two semi-honest MPSI protocols. The first achieves the optimal complexity but is only secure in the augmented semi-honest model. The second protocol, which is secure in the standard semi-honest model, incurs the clients' complexity that depends on the corruption threshold t. Garimella et al. [28] improved these protocols using oblivious key-value store (OKVS) [43, 28, 48, 4] and showed that the first protocol actually supports malicious security. Building on this, Nevo et al. [41] further improved the malicious MPSI protocol in the setting $t < m - 1$.

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. Both protocols' computation complexity is $O(mn)$ per party. Ben-Efraim et al. [3] extended the augmented semi-honest one to the malicious model.

Recently, Wu et al. [61] proposed two semi-honest MPSI protocols based on OPRF and OKVS. These protocols show much better performance than previous works.

MPSI-card and MPSI-card-sum. Chen et al. [15] 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 standard semi-honest model.

MPSU. Kisser and Song [36] introduced the first MPSU protocol. The substantial number of AHE operations and high-degree polynomial calculations in this protocol incur unacceptable efficiency.

Gao et al. [25] proposed a semi-honest MPSU protocol based on public-key operations, which has super-linear complexity in n.

Recently, Dong et al. [17] 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]. 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. We use $[x] = (x_1, \cdots, x_m)$ to denote an additive secret-sharing among m parties, i.e., each P_i holds a share x_i s.t. $x_1 + \cdots x_m = x$. $x \| y$ denotes the concatenation of two strings. We use λ, σ as the computational and statistical security parameters respectively, and use $\overset{s}{\approx}$ (resp.$\overset{c}{\approx}$) to denote that two distributions are statistically (resp. computationally) indistinguishable. For a vector $\mathbf{a}$, a_i denotes the i-th component, $\mathsf{HW}(\mathbf{a})$ denotes hamming weight of $\mathbf{a}$, $\mathsf{zero}(\mathbf{a})$ denotes number of 0 in $\mathbf{a}$, and $\pi(\mathbf{a}) = (a_{\pi(1)}, \cdots, a_{\pi(n)})$, where π is a permutation over n items. The notation $\mathbf{a} \oplus \mathbf{b}$ denotes a component-wise XOR, i.e., $(a_1 \oplus b_1, \cdots, a_n \oplus b_n)$.

2.2 Security Model

In this work, we consider a semi-honest and static adversary $\mathcal{A}$ with the capability to corrupt an arbitrary subset of $t < m$ parties. Let $f = (f_1, \cdots, f_m)$ be a probabilistic polynomial-time m-ary functionality and Π be a m-party protocol for computing f. The view of P_i $(1 \leq i \leq m)$ during an execution of Π on all parties' inputs $\mathbf{x} = (x_1, \cdots, x_m)$ is denoted by $\mathsf{View}_i^{\Pi}(\mathbf{x})$. The output of P_i during an execution of Π on $\mathbf{x}$ is denoted by $\mathsf{Output}_i^{\Pi}(\mathbf{x})$. The joint output of parties is $\mathsf{Output}^{\Pi}(\mathbf{x}) = (\mathsf{Output}_1^{\Pi}(\mathbf{x}), \cdots, \mathsf{Output}_m^{\Pi}(\mathbf{x}))$.

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

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

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

2.3 Multi-party Private Set Operations

MPSO is a special case of MPC. Figure 4 formally defines the typical ideal functionalities computing the intersection, intersection cardinality, intersection sum with cardinality, union, and union cardinality over the parties' private sets.

2.4 Random Oblivious Transfer

Oblivious transfer (OT) [47] is a foundational primitive in MPC, 1-out-of-2 random OT (ROT) functionality $\mathcal{F}_{\mathsf{ROT}}$ is given in Figure 10.

2.5 Batch Oblivious Programmable Pseudorandom Function

The (batch) oblivious programmable pseudorandom function (OPPRF) [38, 45, 12, 54, 48] is an extension of (batch) oblivious pseudorandom function (OPRF) [21, 37, 10]. It allows the sender $\mathcal{S}$ to program a PRF F to output specific uniform values for certain inputs and pseudorandom values for others, while the receiver $\mathcal{R}$ evaluates F without knowledge of the key or the type of output. Detailed descriptions are provided in Appendix A.2, together with the formal batch OPPRF functionality $\mathcal{F}_{\mathsf{bOPPRF}}$ in Figure 11.

Parameters: m parties $P_1, \cdots, P_m$, where P_1 is the leader. Size n of input sets. The bit length l of set elements.
Functionality: On input $X_i = \{x_i^1, \cdots, x_i^n\} \subseteq \{0,1\}^l$ and $V_i = \{v_i^1, \cdots, v_i^n\}$ (with a mapping function $\mathsf{payload}_i()$ from each element to its associated payload) from P_i,

- **MPSI.** give P_1 the intersection $\bigcap_{i=1}^m X_i$.
- **MPSI-card.** give P_1 the intersection cardinality $|\bigcap_{i=1}^m X_i|$.
- **MPSI-card-sum.** for $1 \leq i \leq m$, give each P_i the intersection cardinality $|\bigcap_{i=1}^m X_i|$, and give P_1 the payload sum of the intersection $\sum_{x \in \bigcap_{i=1}^m X_i, 1 \leq j \leq m} \mathsf{payload}_j(x)$.
- **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. 4. Typical Functionalities in MPSO

2.6 Batch Secret-Shared Private Membership Test

Batch secret-shared private membership test (batch ssPMT) [17] is a two-party protocol that implements multiple instances of ssPMT[12, 40] between a sender $\mathcal{S}$ and a receiver $\mathcal{R}$. Given a batch size of B, $\mathcal{S}$ inputs B sets $X_1, \cdots, X_B$, while $\mathcal{R}$ inputs B elements $x_1, \cdots, x_B$. As a result, $\mathcal{S}$ and $\mathcal{R}$ receive secret shares of a bit vector of size B, where the i-th bit is 1 if $x_i \in X_i$, 0 otherwise. The batch ssPMT functionality $\mathcal{F}_{\mathsf{bssPMT}}$ is given in Figure 12 (see Appendix A.3).

2.7 Multi-Party Secret-Shared Shuffle

Multi-party secret-shared shuffle functionality works by randomly permuting the share vectors of all parties and refreshing all shares, ensuring the permutation private to any coalition of $m-1$ parties. The functionality $\mathcal{F}_{\mathsf{shuffle}}$ is given in Figure 13 (see Appendix A.4).

2.8 Hashing to Bins

The hashing-to-bins technique [46, 44] is used in PSO to ensure that identical items across parties are assigned into bins with identical indices. This alignment is achieved by preprocessing input sets via simple hashing and Cuckoo hashing [42].

We denote simple hashing with the following notation:

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

This expression represents hashing items in X into B bins using simple hashing with hash functions $h_1, h_2, h_3 : \{0,1\}^* \to \{1, \cdots, B\}$. The output is a simple hash table denoted by $\mathcal{T}^1, \cdots, \mathcal{T}^B$, where for each $x \in X$ we have $\mathcal{T}^{h_i(x)} \supseteq \{x \| i | i = 1, 2, 3\}$.[8]

We denote Cuckoo hashing with the following notation:

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

This expression represents hashing items in X into B bins using Cuckoo hashing with hash functions h_1, h_2, h_3. The output is a Cuckoo hash table denoted by $\mathcal{C}^1, \cdots, \mathcal{C}^B$, where for each $x \in X$ there is some $i \in \{1, 2, 3\}$ s.t. $\mathcal{C}^{h_i(x)} = \{x \| i\}$.

[8] 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

As discussed in the introduction, our predicate formula representation for any set formulas is designed to satisfy specific structural properties: it is defined as a disjunction of several subformulas, where each subformulas corresponds to one of the disjoint sets Y_i from a partition $\{Y_1, \cdots, Y_s\}$ of the target set Y. Additionally, each Y_i is required to be a subset of some input set X_j. In what follows, we formalize this predicate formula representation by defining a series of notions to capture these structural properties precisely.

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 MPSO. We refer to these resulting sets as constructible sets.

Definition 2. *Let $X_1, \cdots, X_m$ be m sets. A set Y is called a constructible set (over $X_1, \cdots, X_m$) if it can be derived from $X_1, \ldots, X_m$ 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 $X_1, \cdots, X_m$).

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

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

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

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

We prove this theorem in Appendix B.1.

3.2 Canonical Predicate Formula Representation

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

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

where $\psi(x, X_1, \cdots, X_{i-1}, X_{i+1}, \cdots, X_m)$ is a set predicate formula not involving X_i, which we call the separation formula of $\varphi(x, X_1, \cdots, X_m)$ 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 a set predicate formula $\psi(x, X_1, \cdots, X_m)$ be a disjunction of one or more subformulas,[9] denoted as $\psi = Q_1 \vee \cdots \vee Q_s (s \geq 1)$. 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 $\{Y_1, \cdots, Y_s\}$ forms a partition of Y (the constructible set ψ represents), we call $\psi(x, X_1, \cdots, X_m)$ a canonical predicate formula (CPF) representation (over $X_1, \cdots, X_m$).*

[9] A disjunction of one subformulas is itself.

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

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

We prove this theorem in Appendix B.2.

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

Complex set formula. This case has two CPF representations:

- $\psi(x, X_1, X_2, X_3) = ((x \in X_1) \wedge (x \in X_2) \wedge (x \notin X_3)) \vee ((x \in X_1) \wedge (x \in X_3) \wedge (x \notin X_2))$. ψ 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.
- $\psi(x, X_1, X_2, X_3) = (x \in X_1) \wedge [((x \in X_2) \wedge (x \notin X_3)) \vee ((x \in X_3) \wedge (x \notin X_2))]$. ψ 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 to optimize performance.

4 Predicative Zero-Sharing

In this section, we introduce a new notion called predicative zero-sharing, which 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 5.

Parameters: m parties $P_1, \cdots P_m$ with inputs $\mathbf{x} = (x_1, \cdots, x_m)$. 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 $Q(\mathbf{x}) = 1$, $s_1 + \cdots + s_m = 0$. Give s_i to P_i for $1 \leq i \leq m$.

Fig. 5. Predicative Zero-Sharing Functionality $\mathcal{F}_{\mathsf{PZS}}^{Q}$

4.2 Relaxed Predicative Zero-Sharing

Predicative zero-sharing serves as an abstraction of many existing MPC protocols. Given the probabilistic nature of this functionality, a protocol must meet Definition 1 to securely compute predicative zero-sharing. Some protocols, such as multi-party secret-shared ROT (mss-ROT) [17], rigidly conform this definition. However, we observe that a broader range of protocols, such as ROT and equality-conditional randomness generation (ECRG) [35], realize a similar functionality except that if $Q(x) = 0$, the uniform secret shared among the parties is not independent of the joint view of any $m-1$ parties. We demonstrate that ROT implements this relaxed functionality for 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.

To formally capture the security relaxation in this class of protocols, we introduce a simplified security definition for predicative zero-sharing, which comprises three key requirements: correctness, privacy, and independence. We provide detailed definitions of the three requirements and demonstrate its equivalence to the standard security definition through the following theorem.

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

Theorem 3. *If f is a probabilistic functionality in terms of $\boldsymbol{x}$, and Π satisfies:*
- ***Correctness**. The outputs of Π are secret shares of $f(\boldsymbol{x})$, namely,*

$$\{s_1, \cdots, s_m\}_{\boldsymbol{x}} \overset{s}{\approx} \{s_1^\Pi, \cdots, s_m^\Pi\}_{\boldsymbol{x}}$$

- ***Privacy**. There exists a PPT algorithm Sim s.t. for every $\boldsymbol{P}_\mathcal{A} = \{P_{i_1}, \cdots, P_{i_t}\}$,*

$$\{\mathsf{Sim}(\boldsymbol{P}_\mathcal{A}, \boldsymbol{x}_\mathcal{A}, \boldsymbol{s}_\mathcal{A})\}_{\boldsymbol{x}} \overset{c}{\approx} \{\mathsf{View}_\mathcal{A}^\Pi(\boldsymbol{x})\}_{\boldsymbol{x}}$$

- ***Independence**. For every $\boldsymbol{P}_\mathcal{A} = \{P_{i_1}, \cdots, P_{i_t}\}$, the randomness in $f(\boldsymbol{x})$ is independent of $\mathsf{View}_\mathcal{A}^\Pi(\boldsymbol{x})$ during an execution of Π.*

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

$$\{\mathsf{Sim}(\boldsymbol{P}_\mathcal{A}, \boldsymbol{x}_\mathcal{A}, \boldsymbol{s}_\mathcal{A}), s_1, \cdots, s_m\}_{\boldsymbol{x}} \overset{c}{\approx} \{\mathsf{View}_\mathcal{A}^\Pi(\boldsymbol{x}), s_1^\Pi, \cdots, s_m^\Pi\}_{\boldsymbol{x}}$$

We prove this theorem in Appendix C.1.

Remark 1. The predicative zero-sharing functionality $\mathcal{F}_{\mathsf{PZS}}^Q$ 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^Π in the real execution). The independence requirement in this case is instantiated as: if $Q(\mathbf{x}) = 0$, the distribution of the secret $s^\Pi = s_1^\Pi + \cdots + s_m^\Pi$ during an execution of Π is independent of $\mathsf{View}_\mathcal{A}^\Pi(\mathbf{x})$, and the correctness and independence requirements ensure that if $Q(\mathbf{x}) = 0$, s^Π is uniform and independent of the joint view of any $t \le m-1$ parties in real execution.

Note that we use this simplified security definition to prove security for all predicative zero-sharing protocols in this work. Equipped with this security definition, we define relaxed predicative zero-sharing as the form of predicative zero-sharing while only meeting the correctness and privacy requirements. We refer to the relaxed functionality associated with predicate Q as $\mathcal{F}_{\mathsf{rPZS}}^Q$.

4.3 From Relaxed to Standard Predicative Zero-Sharing

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

Assume that all parties have obtained a secret-sharing $[r] = (r_1, \cdots, r_m)$ from a relaxed predicative zero-sharing protocol, with the goal to generate a new secret-sharing $[s] = (s_1, \cdots, s_m)$ 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 this online phase through Beaver triples [2] in Appendix C.2.

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 and independent of the joint view of any $t \leq m-1$ parties. Let E_0 / E_1 be the events $r = 0$ / $r \neq 0$. 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 secure multiplication.

Given that the uniform secrets of the output secret-sharings in relaxed predicative zero-sharing depend on the joint view of some $t \leq m-1$ parties, reconstructions of these secrets after the predicative zero-sharing invocations may potentially leak information in the case of these t parties are in collusion. The transformation technique prevents such leakage by eliminating the dependence. Looking ahead, this step is critical to ensure security against arbitrary collusion in our MPSO framework. Notably, the optimization using Beaver triples employs a star communication topology, allowing it to be integrated into our protocols without altering their structure — thus maintaining their superior complexity while enhancing their security guarantees (see Section 6.2 and 6.3).

4.4 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 6.

Theorem 4. *Protocol Π^Q_{PZS} securely realizes $\mathcal{F}^Q_{\mathsf{PZS}}$ against any semi-honest adversary corrupting $t < m$ parties in the $(\mathcal{F}^{Q_1}_{\mathsf{rPZS}}, \cdots, \mathcal{F}^{Q_q}_{\mathsf{rPZS}})$-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 $\mathcal{F}^{Q'_i}_{\mathsf{rPZS}}$ and $\mathcal{F}^{Q'_j}_{\mathsf{rPZS}}$, $r_i = 0 \wedge r_j = 0$, 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.

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

Protocol:

1. **The simple predicative sharing stage.** In this stage, the parties invoke $\mathcal{F}_{\mathsf{rPZS}}^{Q_i}$ 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. To compute a relaxed predicative zero-sharing for Q', where $Q'(\mathbf{x}) = Q'_i(\mathbf{x}) \wedge Q'_j(\mathbf{x})$, they 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. To compute a relaxed predicative zero-sharing for Q', where $Q'(\mathbf{x}) = Q'_i(\mathbf{x}) \vee Q'_j(\mathbf{x})$, they perform a secure multiplication $[r_i \cdot r_j] = [r_i] \cdot [r_j]$.

   In the end of each step, 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 is the output secret-sharing from the relaxed compound predicative zero-sharing for Q.
3. **Transformation from relaxed to standard.** All parties compute $[s]$ by performing a secure multiplication $[s] = [r] \cdot [b]$ with Beaver triples (c.f. Appendix C.2).

Fig. 6. Predicative Zero-Sharing $\mathit{\Pi}_{\mathsf{PZS}}^{Q}$

– 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 $Pr[E^{i,j}] = Pr[E^{i,j}|E_0^i] \cdot Pr[E_0^i] + Pr[E|E_1^i] \cdot Pr[E_1^i] = 0 \cdot Pr[E_0^i] + 1 \cdot Pr[E_1^i] = Pr[E_1^i] = 1 - Pr[E_0^i]$. To bound the correctness error by $2^{-\sigma}$, we require that the probability of any E_0^i occurring is negligible. By union bound, $Pr[\bigvee_i E_0^i] \leq \sum_i Pr[E_0^i] = \frac{|\mathsf{OR}|}{|\mathbb{F}|}$. Therefore, we set the field size $|\mathbb{F}| \geq |\mathsf{OR}| \cdot 2^{\sigma}$, where $|\mathsf{OR}|$ 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 4.3, 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 6, 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.

5.1 Membership Zero-Sharing

A membership zero-sharing protocol allows m parties ($m \geq 2$), where one party (denoted as P_{pivot}) holds an element x while each of the others P_j holds a set X_j ($j \in \{1, \cdots, m\} \setminus \{\mathsf{pivot}\}$) as input. If the associated set predicate formula $Q(x, X_1, \cdots, X_{\mathsf{pivot}-1}, X_{\mathsf{pivot}+1}, \cdots, X_m) = 1$, they receive secret shares of 0, otherwise they receive secret shares of a random value. A batched version of membership zero-sharing is defined in Figure 7, where P_{pivot} holds a vector $\mathbf{x} = (x_1, \cdots, x_n)$ 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., $\bigwedge_{j \in \{1,\cdots,m\} \setminus \{\mathsf{pivot}\}} x \in X_j$), we refer to it as batch pure membership zero-sharing (tailored for MPSI and its variants in Section 6.2); if Q is a conjunction of $m-1$ set non-membership predicates (i.e., $\bigwedge_{j \in \{1,\cdots,m\} \setminus \{\mathsf{pivot}\}} x \notin X_j$), we refer to it as batch pure non-membership zero-sharing (tailored for MPSU and its variants in Section 6.3). The formal definitions of these two functionalities (together with their constructions) are provided in Appendix D.1 and D.2, respectively. We also introduce a variant of pure membership zero-sharing called pure membership zero-sharing with payloads, where P_{pivot} holds an element x while each of the others holds a set of elements and a set of associated payloads. This functionality is tailored for MPSI-card-sum in Appendix E.2 and its formal definition and construction can be found in Appendix D.3.

Parameters: m parties $P_1, \cdots P_m$, where P_{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 $\mathbf{x} = (x_1, \cdots, x_n)$ from P_{pivot} and $\mathbf{X}_j = (X_{j,1}, \cdots, X_{j,n})$ from each P_j ($j \in \{1, \cdots, m\} \setminus \{\mathsf{pivot}\}$), sample $\mathbf{s}_i = (s_{i,1}, \cdots, s_{i,n}) \leftarrow \mathbb{F}^n$ for $1 \leq i \leq m$, s.t. for $1 \leq d \leq n$, if $Q(x_d, X_{1,d}, X_{\mathsf{pivot}-1,d}, X_{\mathsf{pivot}+1,d}, \cdots, X_{m,d}) = 1$, $\sum_{1 \leq i \leq m} s_{i,d} = 0$. Give $\mathbf{s}_i$ to P_i.

Fig. 7. Batch Membership Zero-Sharing Functionality $\mathcal{F}^{Q}_{\mathsf{bMZS}}$

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 a message leak containing some information about s. Finally, the functionality outputs r, leak to $\mathcal{S}$ and u to $\mathcal{R}$. The leak 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.

From the view of $\mathcal{R}$, if $x \in Y$, $u = -r$; otherwise $u = s - r$ is random as s is uniformly sampled by the functionality and $\mathcal{R}$ receives no additional information about s. Therefore, u can be seen as the value of a "programmable" random function f on x s.t. all function values on elements of Y are programed to

the same uniform $-r$ and other values are random. To efficiently instantiate this functionality, we let f be a programmable PRF. The leakage leak to $\mathcal{S}$ is the PRF key and $\mathcal{R}$ obliviously evaluates $u = f(\mathsf{leak}, x)$ by invoking OPPRF with $\mathcal{S}$. The construction is as follows: $\mathcal{S}$ 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 leak outputted to $\mathcal{S}$ is the PRF key from OPPRF. This protocol can be naturally extended to a batched version 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 a message leak containing some information about s. Finally, the functionality outputs r, leak to $\mathcal{S}$ and u to $\mathcal{R}$.

Intuitively, this functionality shares similarities with the secret-shared private membership test (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 batch ssPMT construction in [17], our goal is to efficiently transform bit secret-sharings into zero-sharings (The batched version proceeds by first having 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, 17], is a relaxed simple predicative zero-sharing with the associated predicate $e_0 \oplus e_1 = 0$. This two-choice-bit ROT actually can be seen as a transformation from bit secret-sharing into zero-sharing: After executing the protocol, $\mathcal{S}$ receives $r_0, r_1 \in \mathbb{F}$ while $\mathcal{R}$ receives $r_{e_0 \oplus e_1} \in \mathbb{F}$[10]. 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 $r + u = r_1 - r_0 = s$ is random. The leakage of s to $\mathcal{S}$ is r_1 (since $\mathcal{S}$ learns r_0, r_1, it can compute s).

5.4 Membership Zero-Sharing for Any Set Predicate Formulas

Using the above instantiations for the two-party relaxed membership zero-sharing protocols and following the technical route in Figure 1, we present the complete protocol of batch membership zero-sharing for any set predicate formula Q in Figure 8.

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

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

[10] 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 8.

Parameters: m parties $P_1, \cdots P_m$, where P_{pivot} holds n elements instead of n sets. A set predicate formula Q composed of $q \geq 1$ literals $Q_1, \cdots, Q_q$ where each Q_i $(1 \leq i \leq q)$ is in the form $x \in X_j$ or $x \notin X_j$ for some $j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$. n Beaver triples $([\mathbf{a}], [\mathbf{b}], [\mathbf{c}])$, where $[\mathbf{a}] = ([a_1], \cdots, [a_n])$, $[\mathbf{b}] = ([b_1], \cdots, [b_n])$, $[\mathbf{c}] = ([c_1], \cdots, [c_n])$ 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_{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. Specifically, if Q_i is in the form
   - $x \in X_j$: 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 $V_{i,k} = \{-r_{i,k}, \cdots, -r_{i,k}\}$, where $|K_{i,k}| = |V_{i,k}|$. Then P_{pivot} and P_j invoke $\mathcal{F}_{\mathsf{bOPPRF}}$ where P_j acts as $\mathcal{S}$ with inputs $(K_{i,1}, \cdots, K_{i,n})$ and $(V_{i,1}, \cdots, V_{i,n})$, and P_{pivot} acts as $\mathcal{R}$ with input $\mathbf{x}$ and receives $\mathbf{u}_i$. P_{pivot} sets its shares $\mathbf{r}_{i,\mathsf{pivot}} = \mathbf{u}_i$. P_j sets its shares $\mathbf{r}_{i,j} = (r_{i,1}, \cdots, r_{i,n})$. For each $d \in \{1, \cdots m\} \setminus \{\mathsf{pivot}, j\}$, P_d sets its shares $\mathbf{r}_{i,d} = \mathbf{0}$.
   - $x \notin X_j$: P_{pivot} and P_j invoke $\mathcal{F}_{\mathsf{bssPMT}}$, where in the k-th instance $(1 \leq k \leq n)$, P_j inputs $X_{j,k}$ and receives $e^0_{i,k}$, while P_{pivot} inputs x_k and receives $e^1_{i,k}$. 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 $r^0_{i,k}, r^1_{i,k}$, while P_{pivot} acts as $\mathcal{R}$ with input $e^1_{i,k}$ and receives $r^{e^1_{i,k}}_{i,k}$. P_{pivot} sets its shares $\mathbf{r}_{i,\mathsf{pivot}} = (r^{e^1_{i,1}}_{i,1}, \cdots, r^{e^1_{i,n}}_{i,n})$. P_j sets its shares $\mathbf{r}_{i,j} = (-r^{e^0_{i,1}}_{i,1}, \cdots, -r^{e^0_{i,n}}_{i,n})$. For each $d \in \{1, \cdots m\} \setminus \{\mathsf{pivot}, j\}$, P_d sets its shares $\mathbf{r}_{i,d} = \mathbf{0}$.

   The vector of n secret-sharings for Q_i denotes as $[\mathbf{r}_i] = (\mathbf{r}_{i,1}, \cdots, \mathbf{r}_{i,m})$.
2. **The formula emulation stage.** The parties follow the same stage as in Figure 6, but generate a vector of n secret-sharings instead of single secret-sharing in each step, where the actions of parties depend on the type of operator:
   - AND operator: Suppose the parties hold two vectors of n secret-sharings $[\mathbf{r}_i]$ and $[\mathbf{r}_j]$. They locally add 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 $[\mathbf{r}_j]$. They perform n secure multiplications between the corresponding components of two vectors, i.e., $[\mathbf{r}_i \cdot \mathbf{r}_j] = [\mathbf{r}_i] \cdot [\mathbf{r}_j]$.

   The ultimate vector $[\mathbf{r}]$ contains n output secret-sharings from the relaxed batch membership zero-sharing for Q.
3. **Transformation from relaxed to standard.** All parties compute $[\mathbf{s}] = [\mathbf{r}] \cdot [\mathbf{b}]$, using $([\mathbf{a}], [\mathbf{b}], [\mathbf{c}])$.

Fig. 8. Batch Membership Zero-Sharing Π^Q_{bMZS}

6 Our MPSO Framework

6.1 Overview

Our framework follows the two high-level phases outlined in the introduction. The first phase based on the CPF representation $\psi(x, X_1, \cdots, X_m)$ of any constructible set Y, is to obtain secret-sharings corresponding to a partition of Y. Recall that $\psi = Q_1 \vee \cdots \vee Q_s$. We use $X_{i_1}, \cdots, X_{i_q}$ $(1 \leq q \leq m)$ to denote all sets relevant to the atomic propositions in Q_i $(1 \leq i \leq s)$, and X_j $(j \in \{i_1, \cdots, i_q\})$ to denote the set which Q_i is set-separable with respect to. Namely,

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

where Q'_i is the separation formula of Q_i. We use Y_i to denote the set represented by Q_i, then Q'_i is the membership testing set predicate formula on elements in X_j with respect to Y_i. In the first phase, for each Q_i, $P_{i_1}, \cdots, P_{i_q}$ invoke n instances of membership zero-sharing associated with Q'_i. In each instance, P_j acts as P_{pivot} inputting one element in X_j. For each $x \in X_j$, if $Q'_i = 1$, the parties receive secret shares of 0, otherwise shares of a random value. P_j adds x to its associated shares, s.t. for each $x \in X_j$, if $Q_i = 1$, i.e., $Q'_i = 1$, the parties receive shares of x, otherwise of a random value.

In the above rough construction, the membership test over X_j requires evaluating Q'_i through n separate membership zero-sharing on identical input sets of size n. As a result, the amortized overhead per instance is $O(n)$ in terms of n, which is inefficient. We optimize this using hashing-to-bins technique to reduce the input set size in each instance. Let P_j preprocess X_j via Cuckoo hashing and the other parties preprocess their sets via simple hashing. The hashing-to-bins technique enables each element $x \in X_j$ to be tested only against the bin with identical indices across all parties, while maintaining correctness of the membership test. Therefore, in each membership zero-sharing instance, when P_j inputs one element from a Cuckoo hash bin, the other parties only need to input the subsets containing elements that are assigned to the corresponding simple hash bins, rather than their entire sets. Since the total size of all input sets across n instances (i.e., the size of simple hash table) is $O(n)$ per party, the amortized overhead per instance is reduced to $O(1)$. The optimized first phase is as follows.

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 $P_{j'}$ $(j' \in \{i_1, \cdots, i_q\} \setminus \{j\})$ 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, s.t. 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

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

$P_{i_1}, \cdots, P_{i_q}$ engage in the batch membership zero-sharing for Q'_i, where P_j acts as P_{pivot} with inputs $\mathcal{C}_j^1, \cdots, \mathcal{C}_j^B$ and each $P_{j'}$ inputs $\mathcal{T}_{j'}^1, \cdots, \mathcal{T}_{j'}^B$. In the end, they receive B secret-sharings, so that if the element x in each bin C_j^b satisfies $Q_i(x, X_{i_1}, \cdots, X_{i_q}) = 1$, 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, so that if $x \in Y_i$, the parties hold a secret-sharing of x.

Given that $\{Y_1, \cdots, Y_s\}$ form a partition of Y, if the parties execute the above process with the last element addition step for all $Q_1, \cdots, Q_s$, 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 $Y_1, \cdots, Y_s$. 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 will instead hold secret-sharings of 0 in the same positions. Whether or not to execute the last step is determined by the target functionality, which can be divided into three categories:

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.

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 MPSO, 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.

Circuit-MPSO. There are two ways to realize circuit-MPSO.

- **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.

6.2 MPSI, MPSI-card and Circuit MPSI

Consider a constructible set Y, which can be represented as a set-separable formula $Q(x, X_1, \cdots, X_m)$ 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 $Q'(x, X_2, \cdots, X_m)$ 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_{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 benefit in this setting is that the costs of the protocols do not scale with the length of 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' 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. The complete protocols are described in Figure 20 (see Appendix E.1). Additionally, our MPSI-card-sum protocol is outlined in Figure 21 (see Appendix E.2). Notably, in naive (insecure) solutions, clients directly sends their input sets to the leader, who computes the result locally — forming a star communication topology. However, prior state-of-the-art protocols [61, 15] require pairwise online interactions among parties. Our protocols introduce a different paradigm that employs the star communication topology, leading to the optimal online complexity: the leader's computation and communication complexity is $O(mn)$ and each client's is $O(n)$, that matches the naive solutions (see Appendix F.1 for detailed complexity analyses). To our knowledge, this is the first time for both MPSI and MPSI-card/MPSI-card-sum to achieve this complexity. In comparison, the state-of-the-art MPSI [61] has a leader's complexity of $O(mn)$ and a client's complexity of $O(tn)$; while the state-of-the-art MPSI-card/MPSI-card-sum [15] has a leader's complexity of $O(mn + tn\log n)$ and a client's complexity of $O(tn)$. Both protocols' client complexity depends on the corruption threshold t, because of their pairwise interaction topology.

6.3 MPSU, MPSU-card and Circuit MPSU

Consider a constructible set Y, whose CPF representation $\psi(x, X_1, \cdots, X_m)$ is a disjunction of several subformulas, one is an atomic proposition $x \in X_i$ for some $1 \leq i \leq m$. For instance, $X_1 \cup \cdots \cup X_m$ can be represented as $(x \in X_1) \vee ((x \notin X_1) \wedge (x \in X_2)) \vee \cdots \vee ((x \notin X_1) \wedge \cdots \wedge (x \notin X_{m-1}) \wedge (x \in X_m))$. In this case, the subformula $x \in X_i$ only involves P_i, so P_i can simply shares its elements. 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. The complete protocols are described in Figure 22 (see Appendix E.3) and their complexity analyses are provided in Appendix F.2. Our MPSU protocol follows the secret-sharing based MPSU paradigm introduced by [40], whose security originally relies on a non-collusion assumption. The prior state-of-the-art MPSU protocol [17] introduces mss-ROT to eliminate the non-collusion assumption, while incurring pairwise online interactions among $P_1, \cdots, P_j$ for each $1 < j \leq m$. This topology of mss-ROT dominates their protocol's complexity, which is $O(m^2 n)$ for both leader and clients. In contrast, our protocol achieves collusion resistance via the transformation technique, whose optimization (using Beaver triples) preserves the star topology among

$P_1, \cdots, P_j$ for each $1 < j \leq m$. As a result, our protocol ensures security against arbitrary collusion while maintaining the original complexity — $O(m^2 n)$ for the leader and $O(mn)$ for each client.

6.4 MPSO, MPSO-card and Circuit MPSO

Let $C^1_{s,B,l}$ and $C^2_{N,l'}$ be two circuits: $C^1_{s,B,l}$ 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 $u_{i,1,b} + \cdots + u_{i,m,b} = 0$ 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; $C^2_{N,l'}$ has 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 $Z = \{z_i | z_{1,i} + \cdots + z_{m,i} = z_i \| 0^{l'}\}_{1\leq i\leq N}$ and f is the function to be computed on Y. The complete MPSO, MPSO-card, and circuit-MPSO protocols for Y are formally presented in Figure 9.

Theorem 6. *The MPSO, MPSO-card and circuit-MPSO protocols in Figure 9 are secure against any semi-honest adversary corrupting $t < m$ parties in the $(\mathcal{F}^Q_{\mathsf{bMZS}}, \mathcal{F}_{\mathsf{shuffle}})$-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 $|\mathbb{F}| \geq |\mathsf{OR}| \cdot B \cdot 2^\sigma$, where $|\mathsf{OR}|$ 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 $\{Y_1, \cdots, Y_s\}$ is a partition of Y, the parties hold $|Y|$ secret-sharings of the elements in Y, and $sB - |Y|$ random secret-sharings in total. Finally, P_1 and $C^2_{sB,l'}$ 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 $B - |Y|$ random secret-sharings. To bound the overall false positive error probability by $2^{-\sigma}$, we set $|\mathbb{F}| \geq sB \cdot 2^\sigma$.

Security. *(Proof sketch)* Similar to the previous proofs for privacy, since all interactions (except for the last reconstruction messages to P_1 in MPSO and MPSO-card) happen within the invocations of blocking blocks — all batch membership zero-sharing protocols and the multi-party secret-shared shuffle, the simulator only needs to emulate these interactions with honest parties by invoking the sub-simulators for these blocking blocks in a backward-chaining manner. The non-trivial part of the simulation is to emulate the last reconstruction messages, in the case P_1 is corrupted. After obtaining Y from the MPSO ideal functionality, the simulator secret-shares the elements in Y as $|Y|$ secret-sharings, then uniformly samples $sB - |Y|$ random secret-sharings. Finally, it randomly permutes all sB secret-sharings. Benefit from the independence requirement of our membership zero-sharing, the uniform secrets of the output secret-sharings are independent of the joint view of any $t \leq m - 1$ parties. Furthermore, the multi-party secret-shared shuffle functionality ensures that any coalition of $t \leq m-1$ parties has no knowledge of the permutation. As a result, the simulation is indistinguishable from the real execution. The full proof is given in Appendix E.4.

In conclusion, we present an framework fully realizing not only the MPSO functionality, but also MPSO-card and circuit-MPSO functionalities. Our framework is primarily based on symmetric-key operations and secure against arbitrary collusion in semi-honest model. The complexity analyses are provided in Appendix F.3.

Parameters. m parties $P_1, \cdots, P_m$. Set size n . The element length l. The all-zero string length l'. A field $\mathbb{F}$. An encoding function $\mathsf{code} : \mathbb{F} \to \{0,1\}^{l+l'}$. A constructible set Y represented as a CPF representation $\psi(x, X_1, \cdots, X_m)$. Cuckoo parameters: hash functions h_1, h_2, h_3 and number of bins B.
Inputs. Each party P_i has input $X_i = \{x_i^1, \cdots, x_i^n\} \subseteq \{0,1\}^l$.
1. Hashing to bin. For $1 \le i \le m$, P_i does $\mathcal{C}_i^1, \cdots, \mathcal{C}_i^B \leftarrow \mathsf{Cuckoo}^B_{h_1,h_2,h_3}(X_i)$ and $\mathcal{T}_i^1, \cdots, \mathcal{T}_i^B \leftarrow \mathsf{Simple}^B_{h_1,h_2,h_3}(X_i)$.
2. Single subformula evaluation. Let $\psi = Q_1 \vee \cdots \vee Q_s$. For the i-th subformula $Q_i(x, X_{i_1}, \cdots, X_{i_q})$ in ψ, where $1 \le i \le s, \{i_1, \cdots, i_q\} \subseteq \{1, \cdots, m\}$,

1. If $q = 1$, suppose $i_1 = \cdots = i_q = j$, then $Q_i(x, X_j) = (x \in X_j)$. For $1 \le b \le B$, 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 $j' \in \{1, \cdots, m\} \setminus \{j\}$, $P_{j'}$ sets $s_{i,j',b} = 0$.
2. If $q > 1$, suppose Q_i is set-separable with respect to X_j for some $j \in \{i_1, \cdots, i_q\}$ and $Q_i(x, X_{i_1}, \cdots, X_{i_q}) = (x \in X_j) \wedge Q_i'(x, X_{i_1}, \cdots, X_{j-1}, X_{j+1}, \cdots, X_{i_q})$. The parties invoke $\mathcal{F}_{\mathsf{bMZS}}^{Q_i'}$ where P_j acts as P_{pivot} with inputs $\mathcal{C}_j^1, \cdots, \mathcal{C}_j^B$ and each $P_{j'}$ inputs $\mathcal{T}_{j'}^1, \cdots, \mathcal{T}_{j'}^B$ for $j' \in \{i_1, \cdots, i_q\} \setminus \{j\}$. For $i' \in \{i_1, \cdots, i_q\}$, each $P_{i'}$ receives $\mathbf{s}_{i,i'} = (s_{i,i',1}, \cdots, s_{i,i',B})$, and for $k \in \{1, \cdots, m\} \setminus \{i_1, \cdots, i_q\}$, each P_k sets $s_{i,k,b} = 0$ for $1 \le b \le B$.

The following actions of parties depend on the functionality:
MPSO.
3. For $1 \le i \le s$, $1 \le b \le B$, if $\mathcal{C}_j^b$ is not an empty bin, P_j (the same j as step **2**) computes $s'_{i,j,b} = \mathsf{code}(s_{i,j,b}) \oplus (x \| 0^{l'})$, 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 $1 \le k \le m$, each P_k computes $\mathbf{u}_k \in (\{0,1\}^{l+l'})^{sB}$ as follows: For $1 \le i \le s, 1 \le b \le B$, $u_{k,(i-1)B+b} = s'_{i,k,b}$.
5. For $1 \le k \le m$, P_k inputs $\mathbf{u}_k$ to $\mathcal{F}_{\mathsf{shuffle}}$ and receives $\mathbf{u}'_k$.
6. For $1 < j \le m$, P_j sends $\mathbf{u}'_j$ to P_1. P_1 computes $\mathbf{u}' = \sum_{1<j\le m} \mathbf{u}'_j$ and sets $Y = \emptyset$. For $1 \le i \le sB$, if $u'_i = y \| 0^{l'}$ for some $y \in \{0,1\}^l$, P_1 update $Y = Y \cup \{y\}$. Output Y.
MPSO-card.
3. For $1 \le k \le m$, P_k computes $\mathbf{u}_k \in \mathbb{F}^{sB}$ as follows: For $1 \le i \le s, 1 \le b \le B$, $u_{k,(i-1)B+b} = s_{i,k,b}$.
4. For $1 \le k \le m$, P_k inputs $\mathbf{u}_k$ to $\mathcal{F}_{\mathsf{shuffle}}$ and receives $\mathbf{u}'_k$.
5. For $1 < j \le m$, P_j sends $\mathbf{u}'_j$ to P_1. P_1 outputs $\mathsf{zero}(\sum_{1<j\le m} \mathbf{u}'_j)$.
Circuit-MPSO (Approach 1).
3. All parties invoke an m-party computation with the circuit $C^1_{s,B,l}$. For $1 \le i \le s, 1 \le k \le m$, P_k inputs $s_{i,k,1}, \cdots, s_{i,k,B}$ to the i-th section, and P_j (the same j as step **2**) inputs the elements in $\mathcal{C}_j^1, \cdots, \mathcal{C}_j^B$ in addition.
Circuit-MPSO (Approach 2).
3. For $1 \le i \le s$, $1 \le b \le B$, if $\mathcal{C}_j^b$ is not empty, P_j (the same j as step **2**) computes $s'_{i,j,b} = \mathsf{code}(s_{i,j,b}) \oplus (x \| 0^{l'})$, 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 $C^2_{sB,l'}$. For $1 \le i \le s, 1 \le b \le B, 1 \le k \le m$, P_k takes $s'_{i,k,b}$ as its $((i-1)B+b)$-th input.

Fig. 9. MPSO/MPSO-card/Circuit-MPSO

7 Performance Evaluation

In MPC, generality is often achieved by sacrificing practicality. For example, generic MPC protocols are typically orders of magnitude slower than specialized MPC protocols. Remarkably, our MPSO framework breaks this pattern. Through the implementations for typical instantiations, including MPSI, MPSI-card, MPSI-card-sum and MPSU, we demonstrate that our framework achieves online performance surpassing or matching the specialized state of the art, while retaining full generality for arbitrary set operations.

Our implementations are available at the following link: https://github.com/real-world-cryptography/MPSO. The implementation details and parameter settings can be found in Appendix G. We assume a commonly used setting where Beaver triples are pre-computed offline and stored locally, following the convention of [63, 40]. Therefore, we focus on the online performance of our framework. To ensure a fair comparison, we also focus on the online performance of all the competitions. Specifically, we compare our protocols with the respective state of the art in the standard semi-honest model:

- The state-of-the-art MPSI [61]: This work proposes two protocols, O-Ring and K-Star, with publicly available implementations [62]. We select K-Star for comparison since it is faster than O-Ring with same total communication costs. We report its online performance by running the full protocol and subtracting the offline costs of random vector oblivious linear evaluation (VOLE) generation, setting the corruption threshold $t = m - 1$. We report the leader's running time and the total communication costs of all parties for both their and our MPSI protocols in Table 1.
- The state-of-the-art MPSI-card [15]: 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 [17]: This work proposes two MPSU protocols with publicly available implementation [16]. 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 the total communication costs of all parties in the online phase in Table 4.

To our knowledge, there is no existing implementation or experimental data for MPSI-card-sum. We provide the first 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 WAN settings, we test on two network bandwidths 400 Mbps and 50Mbps, with 80 ms RTT latency. We run all protocols by having each party hold an input set of equal size and use $m - 1$ threads to interact simultaneously with other parties.

The experimental results in Table 1-4 demonstrate that:

- Our MPSI and its variants show superior online performances to the state of the art in all cases. Specifically, our MPSI protocol achieves a $2.4 - 5.2\times$ speedup (LAN), and a $1.1 - 2.0\times$ / $1.1 - 2.6\times$ speedup (400 / 50 Mbps) compared to [61]; our MPSI-card protocol achieves an $18.0 - 63.4\times$ speedup (LAN) compared to the reported data in [15]. In terms of communication costs, our MPSI becomes increasingly competitive as the number of parties grows, with up to a $2.6\times$ improvement; our MPSI-card achieves a $14.0 - 20.3\times$ improvement compared to [15]. The computation and communication costs of our MPSI-card-sum is only double our MPSI approximately, while realizing a richer functionality.
- Our MPSU shows comparable online performances to the state of the art [17], and has the lowest communication costs in all cases. Notably, our protocol becomes increasingly competitive as m grows in WAN settings, and eventually outperforms [17] with a speedup up to $1.7\times$. Since WAN settings are more representative of MPSU applications, our protocol may be more competitive in real-world scenarios.

Table 1. Running time and communication costs for MPSI protocols. m is the number of parties. m input sets are of equal size.

m	protocol	Time (second)															Comm. (MB)				
		LAN					400Mbps					50Mbps									
		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}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
3	[61]	0.48	0.168	0.822	4.227	19.18	2.863	3.309	4.232	7.759	27.32	3.155	3.544	5.149	11.57	42.38	**0.423**	**1.681**	**6.344**	**26.72**	**110.0**
	Ours	**0.013**	**0.047**	**0.227**	**1.533**	**7.945**	**1.540**	**2.070**	**2.912**	**5.136**	**16.42**	**1.576**	**2.231**	**3.983**	**10.43**	**38.63**	0.641	2.551	10.20	40.91	164.2
4	[61]	0.054	0.181	0.854	4.306	20.09	3.137	3.537	4.528	8.529	30.47	3.158	4.024	6.495	15.90	61.41	**0.896**	**3.557**	**13.42**	**56.11**	**231.1**
	Ours	**0.015**	**0.051**	**0.242**	**1.632**	**8.133**	**1.944**	**2.476**	**3.339**	**6.345**	**19.23**	**1.984**	**2.729**	**4.848**	**14.04**	**51.31**	0.961	3.826	15.31	61.36	246.2
5	[61]	0.072	0.201	0.948	4.611	21.36	3.038	3.733	4.969	9.031	34.16	3.587	4.430	8.443	23.53	90.40	1.542	6.124	23.10	96.23	396.3
	Ours	**0.016**	**0.053**	**0.260**	**1.724**	**8.255**	**2.348**	**2.889**	**3.770**	**6.789**	**22.04**	**2.424**	**3.144**	**5.930**	**16.98**	**64.59**	**1.282**	**5.102**	**20.41**	**81.81**	**328.3**
10	[61]	0.136	0.373	1.479	6.812	30.89	4.681	5.536	7.178	16.84	65.14	6.928	10.87	25.10	87.19	346.5	7.375	29.30	110.5	456.9	1883
	Ours	**0.026**	**0.075**	**0.354**	**1.910**	**8.898**	**4.363**	**4.915**	**5.769**	**9.423**	**33.38**	**4.496**	**5.766**	**11.01**	**34.13**	**133.3**	**2.884**	**11.48**	**45.92**	**184.1**	**738.7**

Table 2. Running time and communication costs for MPSI-card protocols. The data of [15] originates from their paper for lack of available implementation. Cells with − denote missing data that is not reported. Best results are marked in bold.

m	protocol	Time (second)															Comm. (MB)				
		LAN					400Mbps					50Mbps									
		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}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
3	Ours	**0.015**	**0.052**	**0.237**	**1.581**	**8.050**	**1.784**	**2.800**	**3.681**	**6.498**	**19.50**	**1.836**	**3.028**	**5.202**	**13.40**	**46.89**	**0.761**	**3.035**	**12.15**	**48.76**	**195.8**
4	Ours	**0.016**	**0.054**	**0.252**	**1.684**	**8.300**	**2.267**	**3.780**	**5.005**	**8.565**	**24.88**	**2.331**	**4.109**	**6.905**	**18.56**	**64.89**	**1.122**	**4.472**	**17.91**	**71.83**	**288.4**
5	[15]	0.670	1.789	6.289	31.24	–	–	–	–	–	–	–	–	–	–	–	20.70	94.49	425.6	1894	–
	Ours	**0.018**	**0.056**	**0.270**	**1.735**	**8.616**	**2.753**	**4.747**	**6.076**	**10.42**	**28.87**	**2.872**	**5.134**	**8.819**	**23.76**	**83.80**	**1.482**	**5.909**	**23.66**	**94.90**	**381.0**
10	[15]	1.477	4.503	12.81	95.23	–	–	–	–	–	–	–	–	–	–	–	46.58	212.6	957.7	4262	–
	Ours	**0.026**	**0.071**	**0.375**	**2.001**	**9.226**	**5.174**	**9.578**	**11.93**	**18.43**	**50.33**	**5.346**	**10.56**	**18.07**	**49.45**	**174.6**	**3.285**	**13.09**	**52.42**	**210.2**	**844.0**

Table 3. Running time and communication costs for our MPSI-card-sum protocol, as the first MPSI-card-sum implementation.

m	Time (second)															Comm.(MB)				
	LAN					400Mbps					50Mbps									
	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}	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	2.620	3.987	7.237	20.83	78.99	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	3.207	5.268	9.564	28.56	104.4	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	3.843	6.526	12.10	36.71	133.6	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	6.781	12.76	24.96	78.64	287.1	5.493	21.85	87.37	350.2	1405

Table 4. Running time and communication costs for MPSU protocols. Cells with − denote trials running out of memory.

m	protocol	Time (second)															Comm. (MB)				
		LAN					400Mbps					50Mbps									
		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}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
3	[17]	**0.017**	**0.050**	**0.215**	**1.005**	**4.352**	**3.157**	**3.734**	**4.444**	**9.705**	**33.10**	**3.340**	**4.495**	**9.779**	**32.11**	**118.7**	2.416	9.702	39.87	161.2	652.1
	Ours	0.022	0.068	0.298	1.892	9.607	3.327	3.774	4.878	11.04	37.77	3.535	4.606	10.25	32.50	120.97	**2.414**	**9.695**	**39.84**	**161.1**	**651.6**
4	[17]	**0.023**	**0.071**	**0.286**	**1.393**	**5.645**	**3.976**	**4.618**	**6.507**	**17.10**	**59.21**	**4.270**	6.924	19.00	69.23	267.2	5.653	23.06	93.06	376.0	1520
	Ours	0.029	0.089	0.415	2.345	11.25	3.996	4.820	6.756	17.45	64.04	4.360	**6.609**	**17.85**	**63.63**	**245.0**	**5.083**	**20.77**	**83.83**	**338.9**	**1370**
5	[17]	**0.030**	**0.087**	**0.368**	**1.714**	**7.003**	**4.800**	**5.521**	**8.938**	**25.95**	**95.40**	5.419	10.70	32.81	126.5	500.1	10.69	43.52	175.6	709.1	2865
	Ours	0.039	0.114	0.542	2.796	13.18	4.872	5.705	8.992	26.09	101.6	**5.261**	**9.393**	**28.09**	**105.6**	**416.8**	**8.739**	**35.69**	**144.0**	**582.1**	**2358**
10	[17]	**0.088**	**0.286**	**1.183**	–	–	9.203	14.54	38.31	–	–	18.01	56.11	213.2	–	–	74.68	300.2	1210	–	–
	Ours	0.110	0.337	1.483	**7.167**	–	**8.187**	**12.06**	**30.56**	**105.7**	–	**12.14**	**34.22**	**125.1**	**485.7**	–	**42.40**	**170.2**	**686.8**	**2775**	–

Acknowledgments

We thank the anonymous reviewers for their helpful comments. This work is supported by the National Natural Science Foundation of China (No. 62272269), Scientific Research Innovation Capability Support Project for Young Faculty (No. ZYGXQNJSKYCXNLZCXM-I21), and Taishan Scholar Program of Shandong Province.

References

1. 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)
2. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: CRYPTO 1991. vol. 576, pp. 420–432. Springer (1991)
3. 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)
4. 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)
5. Blanton, M., Aguiar, E.: Private and oblivious set and multiset operations. In: ASIACCS 2012. pp. 40–41. ACM (2012)
6. 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: CCS 2019. pp. 291–308. ACM (2019)
7. 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)
8. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 868–886. Springer (2012)
9. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Innovations in Theoretical Computer Science 2012. pp. 309–325. ACM (2012)
10. Bui, D., Couteau, G.: Improved private set intersection for sets with small entries. In: Public-Key Cryptography - PKC 2023. Lecture Notes in Computer Science, vol. 13941, pp. 190–220. Springer (2023)
11. 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)
12. Chandran, N., Gupta, D., Shah, A.: Circuit-psi with linear complexity via relaxed batch OPPRF. Proc. Priv. Enhancing Technol. (2022)
13. Chase, M., Ghosh, E., Poburinnaya, O.: Secret-shared shuffle. In: Advances in Cryptology - ASIACRYPT 2020. Lecture Notes in Computer Science, vol. 12493, pp. 342–372. Springer (2020)
14. 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)
15. Chen, Y., Ding, N., Gu, D., Bian, Y.: Practical multi-party private set intersection cardinality and intersection-sum under arbitrary collusion. In: Inscrypt 2022. Lecture Notes in Computer Science, Springer (2022)
16. Dong, M., Bai, Y., Cao, Y.: MPSU: An efficient c++ repo implementing MPSU" (2025), https://github.com/real-world-cryptography/MPSU
17. Dong, M., Zhang, C., Bai, Y., Chen, Y.: Efficient multi-party private set union without non-collusion assumptions. In: 34rd USENIX Security Symposium, USENIX Security 2025. USENIX Association (2025)
18. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**, 469–472 (1985)
19. Eskandarian, S., Boneh, D.: Clarion: Anonymous communication from multiparty shuffling protocols. In: NDSS 2022. The Internet Society (2022)
20. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. p. 144 (2012), http://eprint.iacr.org/2012/144
21. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: 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: 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. Cryptology ePrint Archive, Paper 2023/1930 (2023), https://eprint.iacr.org/archive/2023/1930/20240316:210303, version: 20240316:210303, https://eprint.iacr.org/archive/2023/1930/20240316:210303
25. Gao, J., Nguyen, S., Trieu, N.: Toward A practical multi-party private set union. Proc. Priv. Enhancing Technol. **2024**(4), 622–635 (2024)
26. Gao, J., Trieu, N., Yanai, A.: Multiparty private set intersection cardinality and its applications. Proc. Priv. Enhancing Technol. **2024**(2) (2024)
27. Gao, Y., Luo, Y., Wang, L., Liu, X., Qi, L., Wang, W., Zhou, M.: Efficient scalable multi-party private set intersection(-variants) from bicentric zero-sharing. In: CCS 2024. ACM (2024)
28. 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)
29. 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)
30. Giorgi, P., Laguillaumie, F., Ottow, L., Vergnaud, D.: Fast secure computations on shared polynomials and applications to private set operations. In: ITC 2024. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024)
31. Hazay, C., Lindell, Y.: Efficient Secure Two-Party Protocols - Techniques and Constructions. Information Security and Cryptography, Springer (2010)
32. Hazay, C., Venkitasubramaniam, 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., Venkitasubramaniam, 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: 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: USENIX Security 2024 (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: 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 - 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: ASIACRYPT 2023. Springer (2023)
41. Nevo, O., Trieu, N., Yanai, A.: Simple, fast malicious multiparty private set intersection. In: 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: EUROCRYPT 2020. Springer (2020)
44. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: Private set intersection using permutation-based hashing. In: USENIX Security 2015. pp. 515–530. USENIX Association (2015)
45. Pinkas, B., Schneider, T., Tkachenko, O., Yanai, A.: Efficient circuit-based PSI with linear communication. In: 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: USENIX Security 2014. pp. 797–812 (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. ACM (2022)
49. Raghuraman, S., Rindal, P., Tanguy, T.: Expand-convolute codes for pseudorandom correlation generators from LPN. In: CRYPTO 2023. Lecture Notes in Computer Science, vol. 14084, pp. 602–632. Springer (2023)
50. Rindal, P.: cryptoTools. (2021), https://github.com/ladnir/cryptoTools
51. Rindal, P.: Coproto. (2024), https://github.com/Visa-Research/coproto
52. Rindal, P.: libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. (2024), https://github.com/osu-crypto/libOTe

53. Rindal, P.: Vole-PSI (2024), https://github.com/Visa-Research/volepsi
54. Rindal, P., Schoppmann, P.: VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In: EUROCRYPT 2021. vol. 12697, pp. 901–930. Springer (2021)
55. Roy, L.: Softspokenot: Quieter OT extension from small-field silent VOLE in the minicrypt model. In: CRYPTO 2022. Springer (2022)
56. 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)
57. Sang, Y., Shen, H., Tan, Y., Xiong, N.: Efficient protocols for privacy preserving matching against distributed datasets. In: ICICS 2006. Lecture Notes in Computer Science, vol. 4307 (2006)
58. 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. pp. 398–412. Springer (2012)
59. Su, J., Chen, Z.: Secure and scalable circuit-based protocol for multi-party private set intersection. CoRR **abs/2309.07406** (2023)
60. 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
61. Wu, M., Yuen, T.H., Chan, K.Y.: O-ring and k-star: Efficient multi-party private set intersection. In: USENIX Security 2024 (2024)
62. Wu, M., Yuen, T.H., Chan, K.Y.: oring (2024), https://github.com/private-panda/oring
63. Zhang, C., Chen, Y., Liu, W., Zhang, M., Lin, D.: Optimal private set union from multi-query reverse private membership test. In: USENIX 2023. pp. 337–354. USENIX Association (2023)
64. Zhang, S.: Efficient VOLE based multi-party PSI with lower communication cost. IACR Cryptol. ePrint Arch. p. 1690 (2023)

A Preliminaries Appendix

A.1 Random Oblivious Transfer

The functionality $\mathcal{F}_{\mathsf{ROT}}$ is formally described in Figure 10. We use SoftSpokenOT [55] to instantiate ROT in this work.

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. 10. 1-out-of-2 Random OT Functionality $\mathcal{F}_{\mathsf{ROT}}$

A.2 Batch Oblivious Programmable Pseudorandom Function

Oblivious pseudorandom function (OPRF) [21, 14, 54] 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 $\mathsf{PRF}(k_i, x_i)$.

Oblivious programmable pseudorandom function (OPPRF) [38, 45, 12, 54, 48] is an extension of OPRF, which lets $\mathcal{S}$ first generate a uniformly random key k and a "hint" hint according to k and a given set of key-value pairs (we denote the hint generation algorithm as $\mathsf{hint} \leftarrow \mathsf{Hint}(k, K, V)$ where K is the set of keys and V is the set of values), then program a PRF F using hint, so that F has specific uniform outputs (values in V) for some specific inputs (keys in K) 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 using its inputs and the given "hint" hint without the knowledge of whether it learns a programmed output of F or just a pseudorandom value. The batch OPPRF functionality $\mathcal{F}_{\mathsf{bOPPRF}}$ is given in Figure 11.

Parameters. Sender $\mathcal{S}$. Receiver $\mathcal{R}$. Batch size B. The bit length l of keys. The bit length γ of values. A PPRF $F : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^l \to \{0,1\}^\gamma$.
Sender's inputs. $\mathcal{S}$ inputs B sets of key-value pairs including:
– Disjoint key sets $\mathbf{K} = (K_1, \cdots, K_B)$.
– Value sets $\mathbf{V} = (V_1, \cdots, V_B)$, where $|K_i| = |V_i|$, $1 \le i \le B$.
Receiver's inputs. $\mathcal{R}$ inputs B queries $\mathbf{x} \subseteq (\{0,1\}^l)^B$.
Functionality: On input $\mathbf{K}$ and $\mathbf{V}$ from $\mathcal{S}$ and $\mathbf{x}$ from $\mathcal{R}$,
– Generate uniform keys $\mathbf{k} = (k_1, \cdots, k_B)$ and $\mathsf{hint} \leftarrow \mathsf{Hint}(\mathbf{k}, \mathbf{K}, \mathbf{V})$ s.t. $F(k_i, \mathsf{hint}, K_i(j)) = V_i(j)$ for $1 \le i \le B, 1 \le j \le |K_i|$;
– Define $f_i = F(k_i, \mathsf{hint}, x_i)$, for $1 \le i \le B$;
– Give $\mathbf{k}$ to $\mathcal{S}$ and give $\mathbf{f} = (f_1, \cdots, f_B)$ and hint to $\mathcal{R}$.

Fig. 11. Batch OPPRF Functionality $\mathcal{F}_{\mathsf{bOPPRF}}$

A.3 Batch Secret-Shared Private Membership Test

The batch ssPMT functionality $\mathcal{F}_{\mathsf{bssPMT}}$ is given in Figure 12. Dong et al. [17] proposed an efficient construction with linear complexity, based on batch OPPRF and secret-shared private equality test (ssPEQT) [45, 12].

Parameters. Sender $\mathcal{S}$. Receiver $\mathcal{R}$. Batch size B. The bit length l of set elements.
Inputs. $\mathcal{S}$ inputs B disjoint sets $X_1, \cdots, X_B$ and $\mathcal{R}$ inputs $\mathbf{x} \subseteq (\{0,1\}^l)^B$.
Functionality. On inputs $X_1, \cdots, X_B$ from $\mathcal{S}$ and input $\mathbf{x}$ from $\mathcal{R}$, for $1 \le i \le B$, 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 $\mathbf{e}_S = (e_S^1, \cdots, e_S^B)$ to $\mathcal{S}$ and $\mathbf{e}_R = (e_R^1, \cdots, e_R^B)$ to $\mathcal{R}$.

Fig. 12. Batch ssPMT Functionality $\mathcal{F}_{\mathsf{bssPMT}}$

A.4 Multi-Party Secret-Shared Shuffle

The formal functionality of multi-party secret-shared shuffle is given in Figure 13. Eskandarian et al. [19] 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 $P_1, \cdots P_m$. The dimension of vector N. The item length l.
Functionality. On input $\mathbf{x}_i = (x_i^1, \cdots, x_i^n)$ from each P_i, sample a random permutation $\pi : [N] \to [N]$. For $1 \le i \le m$, 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. 13. Multi-Party Secret-Shared Shuffle Functionality $\mathcal{F}_{\mathsf{shuffle}}$

A.5 Complexity of Building Blocks

Table 5 summarizes the computation and communication complexities of the building blocks discussed above, for both the online and offline phases.

Table 5. Asymptotic communication and computation costs of building blocks. We use λ, σ as the computational and statistical security parameters respectively. k denotes the field bits in SoftSpokenOT. γ denotes the output length of batch OPPRF. m, N, and l are the number of parties, the dimension of vector, and the item length in multi-party secret-shared shuffle. Different share correlation generation methods result in different offline complexity of multi-party secret-shared shuffle. 'BGV" denotes that the parties generate share correlations using AHE with distributed decryption, where AHE is instantiated with BGV encryption [9]. "ST" denotes that the parties generate share correlations by running pairwise instances of the Share Translation protocol [13] of size N and length l.

<table>
<tr><th rowspan="2">Building block</th><th rowspan="2">Number of parties</th><th colspan="4">online</th><th colspan="4">offline</th><th colspan="2">Share correlation generation</th></tr>
<tr><th colspan="2">computation</th><th colspan="2">communication</th><th colspan="2">computation</th><th colspan="2">communication</th><th colspan="2"></th></tr>
<tr><td>Batch OPPRF [48]</td><td>2</td><td colspan="2">$O(n)$</td><td colspan="2">$O(n\gamma)$</td><td colspan="2">$O(2^k n)$</td><td colspan="2">$O(n\lambda\ k)$</td><td colspan="2" rowspan="3">–</td></tr>
<tr><td>SoftSpokenOT [55]</td><td>2</td><td colspan="2">$O(n)$</td><td colspan="2">$O(n\lvert\mathbb{F}\rvert)$</td><td colspan="2">$O(2^k n)$</td><td colspan="2">$O(n\lambda\ k)$</td></tr>
<tr><td>Batch ssPMT [17]</td><td>2</td><td colspan="2">$O(n)$</td><td colspan="2">$O(n\gamma)$</td><td colspan="2">$O(2^k n\gamma)$</td><td colspan="2">$O(n\lambda\gamma\ k)$</td></tr>
<tr><td rowspan="3">Multi-party secret-shared shuffle [19]</td><td rowspan="3">m</td><td>Leader</td><td>Client</td><td>Leader</td><td>Client</td><td>Leader</td><td>Client</td><td>Leader</td><td>Client</td><td colspan="2"></td></tr>
<tr><td rowspan="2">$O(mN)$</td><td rowspan="2">$O(N)$</td><td rowspan="2">$O(mNl)$</td><td rowspan="2">$O(Nl)$</td><td>$O(mN)$</td><td>$O(N)$</td><td>$O(mNl)$</td><td>$O(Nl)$</td><td>BGV</td><td>BGV</td></tr>
<tr><td colspan="2">$O(mN\log N)$</td><td colspan="2">$O(mN\lambda\log N)$</td><td>ST</td><td>ST</td></tr>
</table>

B Predicate Formula Representation Appendix

B.1 The Proof of Theorem 1

Proof. We prove this theorem by constructing the predicate formula φ using mathematical induction.

- **Base Case.** If $Y = X_i$ for some $i \in \{1, \cdots, m\}$, then $\varphi(x, X_1, \cdots, X_m) = M(x, X_i) : x \in X_i$.
- **Induction Hypothesis.** Assume that for any sets A and B obtained from $X_1, \cdots, X_m$ through k set operations, there exist set predicate formulas φ_A and φ_B such that

$$x \in A \iff \varphi_A(x, X_1, \cdots, X_m) = 1$$

,

$$x \in B \iff \varphi_B(x, X_1, \cdots, X_m) = 1.$$

- **Induction Step.** We proceed to construct φ 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 $\varphi(x, X_1, \cdots, X_m) = \varphi_A(x, X_1, \cdots, X_m) \vee \varphi_B(x, X_1, \cdots, X_m)$.
  2. **Intersection.** If $Y = A \cap B$, $\varphi(x, X_1, \cdots, X_m) = \varphi_A(x, X_1, \cdots, X_m) \wedge \varphi_B(x, X_1, \cdots, X_m)$.
  3. **Difference.** If $Y = A \setminus B$, $\varphi(x, X_1, \cdots, X_m) = \varphi_A(x, X_1, \cdots, X_m) \wedge \neg\varphi_B(x, X_1, \cdots, X_m)$.

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

B.2 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

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

where each C_i $(1 \leq i \leq n)$ is a conjunctive clause. We now show that φ can be transform into another DNF formula ψ with s conjunctive clauses $(s > n)$ s.t. 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:** $C_i = (x \notin X_{i_1}) \wedge \cdots \wedge (x \notin X_{i_t})$ where $t < m$. As C_i is a conjunctive clause of φ, 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

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

  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

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

  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

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

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

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

Let $\psi_1(x, X_1, \cdots, X_m) = C_2 \vee \cdots \vee C_s$, then we have

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

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

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

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

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

We denote as $\psi(x, X_1, \cdots, X_m) = C_1' \vee C_2' \cdots \vee C_s'$. 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.

C Predicative Zero-Sharing Appendix

C.1 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: $\mathbf{t = m - 1}$**

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

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

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 $\mathsf{Sim}(\mathbf{P}_\mathcal{A}, \mathbf{x}_\mathcal{A}, \mathbf{s}_\mathcal{A})$. Meanwhile, by the independence requirement, $\mathbf{r}^{\Pi}$ is independent of $\mathsf{View}_\mathcal{A}^{\Pi}(\mathbf{x})$ in the real execution, so we can obtain

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

We further extend the indistinguishability into

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

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

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^{Π} respectively and obtain

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

The indistinguishability holds because s_h (resp. s_h^{Π}) 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

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

Namely, Π 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: $\mathbf{t = m - k}$**

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

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

where $\mathbf{P}_{\mathcal{H}}$ is the set of honest parties, $\mathsf{View}^{\Pi}_{\mathcal{H}}(\mathbf{x})$ denotes the joint view of $\mathbf{P}_{\mathcal{H}}$, while $\mathbf{s}_{\mathcal{H}}$ and $\mathbf{s}^{\Pi}_{\mathcal{H}}$ 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 $\{\mathsf{Sim}(\{P_1,\cdots,P_m\}\setminus \mathbf{P}_{\mathcal{H}}, \{x_1,\cdots,x_m\}\setminus \mathbf{x}_{\mathcal{H}}, \{s_1,\cdots,s_m\}\setminus \mathbf{s}_{\mathcal{H}}), \{s_1,\cdots,s_m\}\setminus \mathbf{s}_{\mathcal{H}}\}$. Thus, we can conclude that for any subset of parties $\mathbf{P}_{\mathcal{H}} \subset \{P_1,\cdots,P_m\}$ of size k, $\mathbf{s}^{\Pi}_{\mathcal{H}}$ is independent of the joint distribution $\{\{\mathsf{View}^{\Pi}_1(\mathbf{x}),\cdots,\mathsf{View}^{\Pi}_m(\mathbf{x})\}\setminus \mathsf{View}^{\Pi}_{\mathcal{H}}(\mathbf{x}), \{s^{\Pi}_1,\cdots,s^{\Pi}_m\}\setminus \mathbf{s}^{\Pi}_{\mathcal{H}}\}$ in the real execution.

– **Inductive Step: $\mathbf{t = m - k - 1}$**

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

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

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

By the correctness, we also have

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

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

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

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 $\{\mathsf{View}^{\Pi}_{\mathcal{A}}(\mathbf{x}), \mathsf{View}^{\Pi}_{\mathcal{H}}(\mathbf{x})\}$. As $\mathbf{s}^{\Pi}_{\mathcal{A}}$ and $\mathbf{s}^{\Pi}_{\mathcal{H}}$ can be determined by $\mathsf{View}^{\Pi}_{\mathcal{A}}(\mathbf{x})$ and $\mathsf{View}^{\Pi}_{\mathcal{H}}(\mathbf{x})$ respectively, $\mathbf{r}^{\Pi}$ is independent of $\{\mathsf{View}^{\Pi}_{\mathcal{A}}(\mathbf{x}), \mathbf{s}^{\Pi}_{\mathcal{A}}, \mathbf{s}^{\Pi}_{\mathcal{H}}\}$. Furthermore, in the ideal execution, $\mathbf{r}$ is independent of $\{\mathsf{Sim}(\mathbf{P}_{\mathcal{A}}, \mathbf{x}_{\mathcal{A}}, \mathbf{s}_{\mathcal{A}}), \mathbf{s}_{\mathcal{A}}, \mathbf{s}_{\mathcal{H}}\}$, which can extend the previous indistinguishability into

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

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, $s^{\Pi}_h = -(\sum_{s^{\Pi}_i\in\mathbf{s}^{\Pi}_{\mathcal{A}}} s^{\Pi}_i + \sum_{s^{\Pi}_i\in\mathbf{s}^{\Pi}_{\mathcal{H}}} s^{\Pi}_i) + f(\mathbf{x})$. Thus, we extend the previous distributions by including s_h and s^{Π}_h respectively and obtain

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

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

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

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

C.2 Optimized Transformation Technique in Section 4.3

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 $u = u_1 + \cdots + u_m$ and opens it to all parties. As we can see, the transformation only consumes one Beaver triple generated in the offline phase and requires one opening with $2(m-1)\log_2|\mathbb{F}|$ communication overhead for the leader and $2\log_2|\mathbb{F}|$ communication overhead for each client in the online phase.

D Membership Zero-Sharing Appendix

D.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., $\bigwedge_{j\in\{1,\cdots,m\}\setminus\{\mathsf{pivot}\}} x \in X_j$). The ideal functionality $\mathcal{F}_{\mathsf{bpMZS}}$ is formally described in Figure 14. The complete protocol is given in Figure 15.

Parameters: m parties $P_1, \cdots P_m$, where P_{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 $\mathbf{x} = (x_1, \cdots, x_n)$ from P_{pivot} and $\mathbf{X}_j = (X_{j,1}, \cdots, X_{j,n})$ from each P_j ($j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$), sample $\mathbf{s}_i = (s_{i,1}, \cdots, s_{i,n}) \leftarrow \mathbb{F}^n$ for $1 \le i \le m$, s.t. for $1 \le d \le n$, if $\bigwedge_{j\in\{1,\cdots,m\}\setminus\{\mathsf{pivot}\}}(x_d \in X_{j,d}) = 1$, $\sum_{1\le i\le m} s_{i,d} = 0$. Give $\mathbf{s}_i$ to P_i.

Fig. 14. Batch Pure Membership Zero-Sharing Functionality $\mathcal{F}_{\mathsf{bpMZS}}$

Parameters: m parties $P_1, \cdots P_m$. Batch size n. A field $\mathbb{F}$. n Beaver triples $([\mathbf{a}], [\mathbf{b}], [\mathbf{c}])$ generated in the offline phase, where $[\mathbf{a}] = ([a_1], \cdots, [a_n])$, $[\mathbf{b}] = ([b_1], \cdots, [b_n])$, $[\mathbf{c}] = ([c_1], \cdots, [c_n])$ and $c_i = a_i \cdot b_i$ for $1 \le i \le n$.
Inputs: P_{pivot} inputs a vector $\mathbf{x} = (x_1, \cdots, x_n)$. P_j inputs $\mathbf{X}_j = (X_{j,1}, \cdots, X_{j,n})$ for $j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$.
Protocol:

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

Fig. 15. Batch Pure Membership Zero-Sharing Π_{bpMZS}

Complexity Analysis. In the batch pure membership zero-sharing protocol (Figure 14), the costs of each stage are calculated as follows.

– P_{pivot} executes batch OPPRF of size n with each P_j for $j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$. Suppose that in the subsequent invocations of batch OPPRF, each $|X_{j,i}| = N_{j,i} = O(1)$ for $1 \le i \le n, j \in \{1, \cdots m\} \setminus$

$\{\mathsf{pivot}\}$, 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 OPPRF from batch OPRF and OKVS. By leveraging the technique to amortize communication, the total communication of computing n instances of OPPRF is equal to the total number of items $3n$. Furthermore, we utilize vector oblivious linear evaluation (VOLE) [6, 7, 49] to instantiate batch OPRF and the construction in [48] to instantiate OKVS. This ensures the computation complexity of batch OPPRF of size n to scale linearly with n. Therefore, in this stage, the computation and communication complexity of P_{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_{pivot} as the leader. This requires n opening with $O(mn)$ computation/communication complexity for P_{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_{pivot} are $O(mn)$, while the computation and communication complexity of each P_j are $O(n)$.

D.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., $\bigwedge_{j\in\{1,\cdots,m\}\setminus\{\mathsf{pivot}\}} x \notin X_j$). The ideal functionality $\mathcal{F}_{\mathsf{bpNMZS}}$ is formally described in Figure 16. The complete protocols is given in Figure 17.

Parameters: m parties $P_1, \cdots P_m$, where P_{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 $\mathbf{x} = (x_1, \cdots, x_n)$ from P_{pivot} and $\mathbf{X}_j = (X_{j,1}, \cdots, X_{j,n})$ from each P_j ($j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$), sample $\mathbf{s}_i = (s_{i,1}, \cdots, s_{i,n}) \leftarrow \mathbb{F}^n$ for $1 \le i \le m$, s.t. for $1 \le d \le n$, if $\bigwedge_{j\in\{1,\cdots,m\}\setminus\{\mathsf{pivot}\}} x_d \notin X_{j,d} = 1$, $\sum_{1\le i\le m} s_{i,d} = 0$. Give $\mathbf{s}_i$ to P_i.

Fig. 16. Batch Pure Non-Membership Zero-Sharing Functionality $\mathcal{F}_{\mathsf{bpNMZS}}$

Complexity Analysis. In the batch pure non-membership zero-sharing protocol (Figure 16), the costs of each stage are calculated as follows.

- P_{pivot} executes batch ssPMT of size n with each P_j for $j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$. We utilize the construction in [17] based on batch OPPRF and secret-shared private equality test (ssPEQT) [45, 12], which achieves linear computation and communication complexity. Therefore, in this stage, the computation and communication complexity of P_{pivot} are $O(mn)$, while the computation and communication complexity of each P_j are $O(n)$.
- P_{pivot} acts as $\mathcal{R}$ and executes n instances of ROT with each P_j for $j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$. In the offline phases, P_{pivot} and each P_j generate n instances of random-choice-bit ROT, then in the online phase, P_{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_{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_{pivot} as the leader. This requires n opening with $O(mn)$ computation/communication complexity for P_{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_{pivot} are $O(mn)$, while the computation and communication complexity of each P_j are $O(n)$.

Parameters: m parties $P_1, \cdots P_m$. Batch size n. A field $\mathbb{F}$. n Beaver triples $([\mathbf{a}], [\mathbf{b}], [\mathbf{c}])$ generated in the offline phase, where $[\mathbf{a}] = ([a_1], \cdots, [a_n])$, $[\mathbf{b}] = ([b_1], \cdots, [b_n])$, $[\mathbf{c}] = ([c_1], \cdots, [c_n])$ and $c_i = a_i \cdot b_i$ for $1 \leq i \leq n$.
Inputs: P_{pivot} inputs a vector $\mathbf{x} = (x_1, \cdots, x_n)$. P_j inputs $\mathbf{X}_j = (X_{j,1}, \cdots, X_{j,n})$ for $j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$.
Protocol:

1. P_{pivot} and P_j invoke $\mathcal{F}_{\mathsf{bssPMT}}$ where in the i-th instance $(1 \leq i \leq n)$, P_j inputs $X_{j,i}$ and receives $e^0_{j,i}$, while P_{pivot} inputs x_i and receives $e^1_{j,i}$.
2. P_{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 $r^0_{j,i}, r^1_{j,i}$, while P_{pivot} acts as $\mathcal{R}$ inputting $e^1_{j,i}$ and receives $r^{e^1_{j,i}}_{j,i}$. P_{pivot} sets $\mathbf{r}'_j = (r^{e^1_{j,1}}_{j,1}, \cdots, r^{e^1_{j,n}}_{j,n})$.
3. P_{pivot} sets its shares $\mathbf{r}_{\mathsf{pivot}} = \sum_{j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}} \mathbf{r}'_j$. P_j sets its shares $\mathbf{r}_j = (-r^{e^0_{j,1}}_{j,1}, \cdots, -r^{e^0_{j,n}}_{j,n})$. All parties hold a vector of n secret-sharings $[\mathbf{r}] = ([r_1], \cdots, [r_n])$.
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. 17. Batch Pure Non-Membership Zero-Sharing Π_{bpNMZS}

D.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. Specifically, P_{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 formal definition of batch pure membership zero-sharing with payloads functionality is in Figure 18. Note that the payload-sharing only needs to satisfy the relaxed security definition in Section 4.2.

Parameters: m parties $P_1, \cdots P_m$, where P_{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 $\mathsf{payload}_j()$ from P_j's elements to the associated payloads.
Functionality: On input $\mathbf{x} = (x_1, \cdots, x_n)$ from P_{pivot}, $\mathbf{X}_j = (X_{j,1}, \cdots, X_{j,n})$ and $\mathbf{V}_j = (V_{j,1}, \cdots, V_{j,n})$ from each P_j $(j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\})$, sample $\mathbf{s}_i = (s_{i,1}, \cdots, s_{i,n}) \leftarrow \mathbb{F}^n, \mathbf{w}_i = (s_{w,1}, \cdots, s_{w,n}) \leftarrow \mathbb{F}'^n$ for $1 \leq i \leq m$, s.t. for $1 \leq d \leq n$, if $\bigwedge_{j \in \{1, \cdots, m\} \setminus \{\mathsf{pivot}\}} (x_d \in X_{j,d}) = 1$, $\sum_{1 \leq i \leq m} s_{i,d} = 0$ and $\sum_{1 \leq i \leq m} w_{i,d} = \sum_{j \in \{1, \cdots, m\} \setminus \{\mathsf{pivot}\}} v_{j,d}$, where $v_{j,d} = \mathsf{payload}_j(x_d) \in V_{j,d}$. Give $(\mathbf{s}_i, \mathbf{w}_i)$ to P_i.

Fig. 18. Batch Pure Membership Zero-Sharing with Payloads Functionality $\mathcal{F}_{\mathsf{bpMZSp}}$

The construction of batch pure membership zero-sharing with payloads protocol resembles the batch pure membership zero-sharing protocol in Figure 15. 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_{pivot} holds an element x and each P_j ($j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$) holds an element set X_j and a payload set V_j. P_{pivot} engages in the two-party version with each P_j, where P_{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_{pivot} sets $r_{\mathsf{pivot}} = \sum_{j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}} t_j$ as its first secret share and sets $w_{\mathsf{pivot}} = \sum_{j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}} u_j$ 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, $\sum_{1 \le i \le m} r_i = 0$ and $\sum_{1 \le i \le m} w_i = \sum_{j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}} v_j$, otherwise $\sum_{1 \le i \le m} r_i$ and $\sum_{1 \le i \le m} w_i$ 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 19.

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

- P_{pivot} executes batch OPPRF of size n with each P_j for $j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$. In this stage, the computation and communication complexity of P_{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_{pivot} as the leader. This requires n opening with $O(mn)$ computation/communication complexity for P_{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_{pivot} are $O(mn)$, while the computation and communication complexity of each P_j are $O(n)$.

Parameters: m parties $P_1, \cdots P_m$. Batch size n. A field $\mathbb{F}$ and payload field $\mathbb{F}'$. The mapping function $\mathsf{payload}_j()$ from P_j's elements to the associated payloads. n Beaver triples $([\mathbf{a}], [\mathbf{b}], [\mathbf{c}])$ generated in offline phase, where $[\mathbf{a}] = ([a_1], \cdots, [a_n])$, $[\mathbf{b}] = ([b_1], \cdots, [b_n])$, $[\mathbf{c}] = ([c_1], \cdots, [c_n])$ and $c_i = a_i \cdot b_i$ for $1 \le i \le n$.
Inputs: P_{pivot} inputs a vector $\mathbf{x} = (x_1, \cdots, x_n)$. P_j inputs $\mathbf{X}_j = (X_{j,1}, \cdots, X_{j,n})$ and $\mathbf{V}_j = (V_{j,1}, \cdots, V_{j,n})$ for $j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$.
Protocol:

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

Fig. 19. Batch Pure Membership Zero-Sharing with Payloads Π_{bpMZSp}

E MPSO Appendix

E.1 MPSI, MPSI-card and Circuit MPSI

Let $C^1_{s,B,l}$ 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 ($1 \le i \le s$), the k-th group of B inputs on $\mathbb{F}$ is associated with P_k for $1 \le k \le m$, and we denote the b-th input in this group ($1 \le b \le B$) as $u_{i,k,b} \in \mathbb{F}$; The last B l-length inputs are associated with P_j for certain $1 \le j \le m$, where we denote the b-th input ($1 \le b \le B$) as $z_{i,b} \in \{0,1\}^l$. The circuit first consists a subcircuit producing a bit $w_{i,b} = 1$ if $u_{i,1,b} + \cdots + u_{i,m,b} = 0$ and 0 otherwise for $1 \le i \le s, 1 \le b \le B$. Then, the circuit computes and outputs $f(Z)$ where $Z = \{z_{i,b}|w_{i,b} = 1\}_{1\le i\le s,1\le b\le B}$ and f is the function to be computed on the constructible set Y. The formal description of our MPSI, MPSI-card and circuit-MPSI (Approach 1) protocols are given in Figure 20.

Parameters. m parties $P_1, \cdots, P_m$. 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 $X_i = \{x_i^1, \cdots, x_i^n\} \subseteq \{0,1\}^l$.

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

The following actions of the parties depend on the functionality:
MPSI.

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

MPSI-card.

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

Circuit-MPSI (Approach 1).

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

Fig. 20. MPSI/MPSI-card/Circuit-MPSI

E.2 MPSI-card-sum

The formal description of our MPSI-card-sum protocol is given in Figure 21.

E.3 MPSU, MPSU-card and Circuit MPSU

Let $C^2_{N,l'}$ be a circuit with m groups of N inputs on $\mathbb{F}$. The k-th group is associated with P_k ($1 \le k \le m$), where the i-th inputs is denoted by $z_{k,i}$ ($1 \le i \le N$). The circuit computes and outputs $f(Z)$ where $Z = \{z_i|z_{1,i} + \cdots + z_{m,i} = z_i\|0^{l'}\}_{1\le i\le N}$ and f is the function to be computed on Y. The formal description of our MPSU, MPSU-card, and circuit-MPSU (Approach 2) protocols are given in Figure 22.

Parameters. Same as in Figure 20.
Inputs. Each party P_i has input $X_i = \{x_i^1, \cdots, x_i^n\} \subseteq \{0,1\}^l$ and $V_i = \{v_i^1, \cdots, v_i^n\}$, with a mapping function $\mathsf{payload}_i()$ from each element to its associated payload.

1. **Hashing to bin.** P_1 does $\mathcal{C}_1^1, \cdots, \mathcal{C}_1^B \leftarrow \mathsf{Cuckoo}^B_{h_1,h_2,h_3}(X_1)$. For $1 < j \leq m$, P_j does $\mathcal{T}_j^1, \cdots, \mathcal{T}_j^B \leftarrow \mathsf{Simple}^B_{h_1,h_2,h_3}(X_j)$. P_j defines $\mathcal{V}_j^1, \cdots, \mathcal{V}_j^B$ 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 $\mathcal{F}_{\mathsf{bpMZSp}}$ of batch size B, where P_1 acts as P_{pivot} with inputs $\mathcal{C}_1^1, \cdots, \mathcal{C}_1^B$ and each of the remaining parties P_j inputs $(\mathcal{T}_j^1, \cdots, \mathcal{T}_j^B)$ and $(\mathcal{V}_j^1, \cdots, \mathcal{V}_j^B)$. For $1 \leq i \leq m$, P_i receives $(\mathbf{s}_i, \mathbf{w}_i)$.
3. 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 $\mathcal{C}_1^b$.
4. For $1 \leq i \leq m$, P_i invoke $\mathcal{F}_{\mathsf{shuffle}}$ with input $\mathbf{s}_i$ and $\mathbf{w}_i$, respectively. P_i receives $\mathbf{s}'_i = (s'_1 \cdots, s'_B)$ and $\mathbf{w}'_i = (w'_1 \cdots, w'_B)$.
5. 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 $\mathbf{e} = (e_1, \cdots, e_B)$ 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 $\mathsf{HW}(\mathbf{e})$.
6. For $1 \leq i \leq m$, P_i computes $u_i = \sum_{1\leq b\leq B \text{s.t.} e_b=1} w'_{i,b}$. 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. 21. MPSI-card-sum

E.4 Security Proof for 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 $\mathbf{s}_{i,c}$ from each $\mathcal{F}^{Q'_i}_{\mathsf{bMZS}}$ for $1 \leq i \leq s$ (if P_c belongs to the set of Q_i's involving parties $\{P_{i_1}, \cdots, P_{i_q}\}$), shuffled secret shares $\mathbf{u}'_c$ from $\mathcal{F}_{\mathsf{shuffle}}$, 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 $\mathbf{Q}_{\mathcal{A}} = \{Q_{j_1}, \cdots, Q_{j_{s'}}\} \subseteq \{Q_1, \cdots, Q_s\}$ without involving honest parties, and $\mathbf{Q}_{\mathcal{H}} = \{Q_1, \cdots, Q_s\} \setminus \mathbf{Q}_{\mathcal{A}}$, 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 $\mathcal{F}^{Q'_h}_{\mathsf{bMZS}}$ 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 $\mathcal{F}_{\mathsf{shuffle}}$.
- **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 π and secret-shares the shuffled union as $\mathbf{u}'_1, \cdots, \mathbf{u}'_m$, where $\mathbf{u}'_c$ is outputted to P_c as secret shares from $\mathcal{F}_{\mathsf{shuffle}}$ for each $P_c \in \{P_{i_1}, \cdots, P_{i_q}\}$.

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

$$\{\mathsf{Sim}^{Q_h}(\mathbf{P}^h_{\mathcal{A}}, \mathbf{X}^h_{\mathcal{A}}, \{\mathbf{s}_{h,c}\}_{P_c\in\mathbf{P}^h_{\mathcal{A}}})_{Q_h\in\mathbf{Q}_{\mathcal{H}}}, \{\mathbf{s}_{h,c}\}_{Q_h\in\mathbf{Q}_{\mathcal{H}}, P_c\in\mathbf{P}^h_{\mathcal{A}}}\}\mathbf{X}$$
$$\overset{c}{\approx} \{\mathsf{View}^{Q_h}_{\mathcal{A}}(\mathbf{X}^h)_{Q_h\in\mathbf{Q}_{\mathcal{H}}}, \{\mathbf{s}^{\Pi}_{h,c}\}_{Q_h\in\mathbf{Q}_{\mathcal{H}}, P_c\in\mathbf{P}^h_{\mathcal{A}}}\}\mathbf{X},$$

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

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

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

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 $s_{j,j,b} = \mathsf{code}(s_{j,j,b}) \oplus (x \| 0^{l'})$, 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 $\mathcal{F}_{\mathsf{shuffle}}$ 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 $1 \leq b \leq (m-1)B$, if $u_b' = y \| 0^{l'}$ for some $y \in \{0,1\}^l$, P_1 update $Y = Y \cup \{y\}$. Output 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 $\mathcal{F}_{\mathsf{shuffle}}$ 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 $n + \mathsf{zero}(\sum_{1<j\leq m} \mathbf{u}_j')$.

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 $s_{j,j,b} = \mathsf{code}(s_{j,j,b}) \oplus (x \| 0^{l'})$, 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 $C^2_{(m-1)B,l'}$. For $1 \leq i \leq m, 1 \leq k \leq (m-1)B$, P_i inputs $u_{i,c}$ to the circuit.

Fig. 22. MPSU/MPSU-card/Circuit-MPSU

where $\mathbf{X} = \{X_1, \cdots, X_m\}$. $\mathbf{P}^h_{\mathcal{A}}$ denotes the corrupted parties involving in Q_h, while $\mathbf{X}^h_{\mathcal{A}}$ denotes the set of their inputs sets. $\mathbf{X}^h$ denotes the set of all involved parties' inputs sets in Q_h. Sim^{Q_h} is the view emulated by the simulator of $\mathcal{F}^{Q'_h}_{\mathsf{bMZS}}$, while $\mathsf{View}^{Q_h}_{\mathcal{A}}$ is the real view of adversary in the batch membership zero-sharing protocol for Q_h. The distinctions with a superscript Π 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

$$\begin{aligned}&\{\mathsf{Sim}^{Q_i}(\mathbf{P}^i_{\mathcal{A}}, \mathbf{X}^i_{\mathcal{A}}, \{\mathbf{s}_{i,c}\}_{P_c \in \mathbf{P}^i_{\mathcal{A}}})_{1 \le i \le s}, \{\mathbf{s}_{i,c}\}_{1 \le i \le s, P_c \in \mathbf{P}_{\mathcal{A}}}\}_{\mathbf{X}} \\ \overset{c}{\approx}\ &\{\mathsf{View}^{Q_i}_{\mathcal{A}}(X_{i_1}, \cdots, X_{i_q})_{1 \le i \le s}, \{\mathbf{s}^{\Pi}_{i,c}\}_{1 \le i \le s, P_c \in \mathbf{P}_{\mathcal{A}}}\}_{\mathbf{X}}.\end{aligned}$$

By correctness, after invoking all $\mathcal{F}^{Q'_h}_{\mathsf{bMZS}}$ 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 $R^{\Pi}_{\mathcal{H}}$). By the independence requirement of $\mathcal{F}^{Q'_h}_{\mathsf{bMZS}}$, all random values in $R^{\Pi}_{\mathcal{H}}$ 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 $\mathcal{F}^{Q'_h}_{\mathsf{bMZS}}$. After the execution of multi-party secret-shared shuffle, the order of elements in $Y \cup R^{\Pi}_{\mathcal{H}} \cup R^{\Pi}_{\mathcal{A}}$ is shuffled. By the functionality of $\mathcal{F}_{\mathsf{shuffle}}$, the random permutation π^{Π} is sampled independently. Thereby,

$$\begin{aligned}&\{\mathsf{Sim}^{Q_i}(\mathbf{P}^i_{\mathcal{A}}, \mathbf{X}^i_{\mathcal{A}}, \{\mathbf{s}_{i,c}\}_{P_c \in \mathbf{P}^i_{\mathcal{A}}})_{1 \le i \le s}, \{\mathbf{s}_{i,c}\}_{1 \le i \le s, P_c \in \mathbf{P}_{\mathcal{A}}}, \pi(Y \cup R_{\mathcal{H}} \cup R_{\mathcal{A}})\}_{\mathbf{X}} \\ \overset{c}{\approx}\ &\{\mathsf{View}^{Q_i}_{\mathcal{A}}(X_{i_1}, \cdots, X_{i_q})_{1 \le i \le s}, \{\mathbf{s}^{\Pi}_{i,c}\}_{1 \le i \le s, P_c \in \mathbf{P}_{\mathcal{A}}}, \pi^{\Pi}(Y \cup R^{\Pi}_{\mathcal{H}} \cup R^{\Pi}_{\mathcal{A}})\}_{\mathbf{X}}.\end{aligned}$$

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

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

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

$$\begin{aligned}&\{\mathsf{Sim}^{Q_i}(\mathbf{P}^i_{\mathcal{A}}, \mathbf{X}^i_{\mathcal{A}}, \{\mathbf{s}_{i,c}\}_{P_c \in \mathbf{P}^i_{\mathcal{A}}})_{1 \le i \le s}, \{\mathbf{s}_{i,c}\}_{1 \le i \le s, P_c \in \mathbf{P}_{\mathcal{A}}}, \mathsf{Sim}^{sh}(\mathbf{P}_{\mathcal{A}}, \{\mathbf{u}_c, \mathbf{u}'_c\}_{P_c \in \mathbf{P}_{\mathcal{A}}}), \mathbf{u}'_1, \cdots, \mathbf{u}'_m\}_{\mathbf{X}} \\ \overset{c}{\approx}\ &\{\mathsf{View}^{Q_i}_{\mathcal{A}}(X_{i_1}, \cdots, X_{i_q})_{1 \le i \le s}, \{\mathbf{s}^{\Pi}_{i,c}\}_{1 \le i \le s, P_c \in \mathbf{P}_{\mathcal{A}}}, \mathsf{View}^{sh}_{\mathcal{A}}(\mathbf{u}^{\Pi}_1, \cdots, \mathbf{u}^{\Pi}_m), \mathbf{u}'^{\Pi}_1, \cdots, \mathbf{u}'^{\Pi}_m\}_{\mathbf{X}},\end{aligned}$$

where $\mathbf{u}_k$ is computed by $\{\mathbf{s}_{i,k}\}_{1 \le i \le s}$ 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 $P_1 \in \mathbf{P}_{\mathcal{A}}$.

F Theoretical Analysis and Comparison

F.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 20, the parties (P_1 as P_{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 ($1 < j \le m$) 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 [19] 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 21, 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.

F.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 22, $1 < j \leq m$, $P_1, \cdots, P_j$ (P_j as P_{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_{pivot} in the first time. P_1 engages in $m-1$ invocations of batch pure non-membership zero-sharing protocols without acting as P_{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.

F.3 Complexity of Our MPSO Framework

The computation and communication complexity in Figure 9 are both dominated by the form of the CPF representation $\psi(x, X_1, \cdots, X_m)$ of the constructible set Y being computed, where $\psi = Q_1 \vee \cdots \vee Q_s$.

In the subformula evaluation stage, the computation complexity of each party P_j includes two parts: (1) $O(\sum_{1\leq i\leq s}(i_q \cdot |\mathsf{AND}_i + \mathsf{OR}_i| \cdot n))$, 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 $|\mathsf{AND}_i + \mathsf{OR}_i|$ is the total number of AND and OR operators in Q_i'; (2) $O(\sum_{1\leq i\leq s}(|\mathsf{AND}_i + \mathsf{OR}_i| \cdot n))$, where Q_i is not set-separable with respect to X_j while includes X_j, and $|\mathsf{AND}_i + \overline{\mathsf{OR}_i}|$ 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) $O(\sum_{1\leq i\leq s}(i_q \cdot |\mathsf{OR}_i| \cdot n))$, 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 $|\mathsf{OR}_i|$ is

the number of OR operators in Q_i'; (2) $O(\sum_{1\le i\le s}(|\mathsf{OR}_i|\cdot n))$, where Q_i is not set-separable with respect to X_j while includes X_j, and $|\mathsf{OR}_i|$ 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 ψ.

F.4 Comparison with Prior Works

Table 6 and 7 shows a theoretical comparison of the computation and communication required by various MPSI protocols in the online/offline phase. Table 8 and 9 shows a theoretical comparison between the related MPSI-card/MPSI-card-sum protocols and ours in the online/offline phase. Table 10 and 11 shows a theoretical comparison between the related MPSU protocols and ours in the online/offline phase.

Table 6. Asymptotic online communication and computation costs of MPSI protocols in the semi-honest setting. n is the set size. m is the number of parties. t is the number of colluding parties. U is the domain of elements. 'PK" denotes the protocols based on public-key operations, and "SK" denotes the protocols based on OT and symmetric-key operations. We use λ, σ 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.

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]	$O(mn)$		$O(mn)$	$O(\log(m)n\sigma^2)$	augmented semi-honest	SK
				$O(mn\sigma^2)$	standard semi-honest	SK
[29]	$O(mn\log n)$	$O(n\log n)$	$O((m^2+mn)\lambda)$		malicious	SK
[3]	$O(mn)$		$O(mn\lambda(\log(n\lambda)+\lambda))$	$O(n\lambda(\log(n\lambda)+\lambda))$	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
[61]	$O(mn)$	$O(tn)$	$O(mn(\lambda+\sigma+\log n))$	$O(tn(\lambda+\sigma+\log n))$	standard semi-honest	SK
Ours	$O(mn)$	$O(n)$	$O(mn(\sigma+\log n))$	$O(n(\sigma+\log n))$	standard semi-honest	SK

Table 7. Asymptotic offline communication and computation costs of MPSI protocols in the semi-honest setting. The only prior MPSI work that is divided into offline and online phases is [3]. We omit prior works without the offline phase. Different Beaver triple generation methods result in different offline complexity of our protocol. 'BGV" denotes that the parties generate Beaver triples using AHE with distributed decryption, where AHE is instantiated with BGV encryption [9]. "OT" denotes that the parties generate Beaver triples using pairwise OT executions, where OT is instantiated with SoftSpokenOT [55]. For the sake of simplicity, we omit some small constants, including the ciphertext expansion constant and the field bits in SoftSpokenOT.

Protocol	Computation		Communication		Security	Beaver triple generation
	Leader	Client	Leader	Client		
[3]	$O(mn)$		$O(mn\lambda^2+mn\lambda(\log(n\lambda)+\lambda))$	$O(m\lambda+n\lambda(\log(n\lambda)+\lambda))$	augmented semi-honest/malicious	–
Ours	$O(mn)$	$O(n)$	$O(mn(\lambda+\sigma+\log n))$	$O(n(\lambda+\sigma+\log n))$	standard semi-honest	BGV
	$O(mn(\sigma+\log n))$		$O(mn(\sigma+\log n)\lambda)$		standard semi-honest	OT

Table 8. Asymptotic online 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. t is the number of colluding parties. "SK" denotes the protocols based on OT and symmetric-key operations. We use λ, σ as the computational and statistical security parameters respectively.

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

Table 9. Asymptotic offline communication and computation costs of MPSI-card/MPSI-card-sum protocols in the semi-honest setting. Different combinations of Beaver triple and share correlation generation methods result in different offline complexity of both [15] and our protocols. 'BGV + BGV" denotes that the parties generate both Beaver triples and share correlations using AHE with distributed decryption, where AHE is instantiated with BGV encryption [9]. 'BGV + ST" denotes that the parties generate Beaver triples using AHE with distributed decryption, while generate share correlations by running pairwise instances of the Share Translation protocol [13]. "OT + BGV/ST" denotes that the parties generate Beaver triples using pairwise SoftSpokenOT [55] executions. (In this case, different share correlation generation methods leads to the same complexity.)

Protocol	**Computation**		**Communication**		**Security**	**Beaver triple + Share correlation**
	Leader	Client	Leader	Client		
[15]	$O(mn)$		$O(mn(\sigma+\log n))$		standard semi-honest	BGV + BGV
	$O(mn\log n)$		$O(mn(\sigma+\lambda\log n))$		standard semi-honest	BGV + ST
	$O(mn(\sigma+\log n))$		$O(mn(\sigma+\log n)\lambda)$		standard semi-honest	OT + BGV/ST
Ours	$O(mn)$	$O(n)$	$O(mn(\lambda+\sigma+\log n))$	$O(n(\lambda+\sigma+\log n))$	standard semi-honest	BGV + BGV
	$O(mn\log n)$		$O(mn(\sigma+\lambda\log n))$		standard semi-honest	BGV + ST
	$O(mn(\sigma+\log n))$		$O(mn(\sigma+\log n)\lambda)$		standard semi-honest	OT + BGV/ST

Table 10. Asymptotic online communication and computation costs of MPSU protocols in the semi-honest setting. n is the set size. m is the number of parties. l is the length of elements. 'PK" denotes protocols based on public-key operations. "SK" denotes protocols based on OT and symmetric-key operations. We use λ, σ as the computational and statistical security parameters.

Protocol	**Computation**		**Communication**		**Security**	**Operation**
	Leader	Client	Leader	Client		
[25]	$O(mn(\log n/\log\log n))$		$\lambda mn(\log n/\log\log n)$		standard semi-honest	PK
[17]	$O(m^2n)$		$O(m^2n(l+\sigma+\log m+\log n))$	$O(mn(l+\sigma+\log m+\log n)+m^2n)$	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 11. Asymptotic offline communication and computation costs of MPSU protocols in the semi-honest setting. The protocol in [25] does not have the offline phase. Different combinations of Beaver triple and share correlation generation methods result in different offline complexity of our protocol. 'BGV + BGV" denotes that the parties generate both Beaver triples and share correlations using AHE with distributed decryption, where AHE is instantiated with BGV encryption [9]. 'BGV + ST" denotes that the parties generate Beaver triples using AHE with distributed decryption, while generate share correlations by running pairwise instances of the Share Translation protocol [13]. "OT + BGV/ST" denotes that the parties generate Beaver triples using pairwise SoftSpokenOT [55] executions. (In this case, different share correlation generation methods leads to the same complexity.)

Protocol	**Computation**		**Communication**		**Beaver triple + Share correlation**
	Leader	Client	Leader	Client	
[17]	$O(m^2n+mn(l+\sigma+\log m+\log n))$		$O(mn(l+\sigma+\log m+\log n)(m+\lambda))$		$-+BGV$
	$O(m^2n(\log m+\log n)+mn(l+\sigma))$		$O(m^2n\lambda(\log m+\log n)+mn(l+\sigma)(m+\lambda))$		$-+ST$
Ours	$O(m^2n)$	$O(mn)$	$O(mn(l+\sigma+\log m+\log n)(m+\lambda))$	$O(mn\lambda(l+\sigma+\log m+\log n))$	BGV + BGV
	$O(m^2n(\log m+\log n))$		$O(m^2n\lambda(\log m+\log n)+mn(l+\sigma)(m+\lambda))$	$O(m^2n\lambda(\log m+\log n)+mn\lambda(l+\sigma))$	BGV + ST
	$O(m^2n(l+\sigma+\log m+\log n))$		$O(m^2n(l+\sigma+\log m+\log n)\lambda)$		OT + BGV/ ST

G Implementation Details and Parameter Settings

G.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, 28, 48, 4], following [10]; We instantiate batch ssPMT with batch OPPRF and ssPEQT, following [17]. We use the following libraries in our implementation.

- VOLE: We use VOLE implemented in libOTe [52], instantiating the code family with Expand-Convolute codes [49].
- OKVS and GMW: We use the optimized OKVS construction in [48][11] and re-use the OKVS implementation in [53]. We also re-use the GMW implementation in [53] to construct ssPEQT.
- ROT: We use SoftSpokenOT [55] implemented in libOTe.
- Additionally, we use the cryptoTools [50] library to compute hash functions and PRNG calls, and we adopt Coproto [51] to realize network communication.

G.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 [17], 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.

H MPSO from Predicative Zero-Encryption

In this section, we introduce a new primitive called predicative zero-encryption, which is a variant of predicative zero-sharing protocol based on multi-key rerandomizable public-key encryption (MKR-PKE). This functionality can be relaxed as relaxed predicative zero-encryption, in a similar manner to predicative zero-sharing. The relaxed predicative zero-encryption protocol can be easily constructed from relaxed predicative zero-sharing, thereby inheriting its composability property. The transformation from relaxed predicative zero-encryption into the standard version is realized by applying a new transformation technique (without

[11] Since the existence of suitable parameters for the new OKVS construction of the recent work [4] is unclear when the set size is less than 2^{10}, we choose to use the OKVS construction of [48].

using Beaver triples). However, this technique requires the underlying MKR-PKE to possess a plaintext space that forms a field. The encryption based counterpart of membership zero-sharing — membership zero-encryption — is a particular class of predicative zero-encryption tailored for MPSO. It is constructed following the same technical route: starting from relaxed membership zero-encryption built on relaxed membership zero-sharing, and then applying the transformation technique to achieve the standard version. With membership zero-encryption serving as the main building block, we present an encryption based framework that can also fully realizes the MPSO and MPSO-card functionalities, in addition to the secret-sharing based framework presented in the main body. We conclude the section with a comparison of the two MPSO frameworks, discussing their respective advantages and disadvantages.

H.1 Multi-Key Rerandomizable Public-Key Encryption

Gao et al. [24] introduced multi-key rerandomizable public-key encryption (MKR-PKE) and instantiated it using elliptic curve (EC) based ElGamal encryption [18]. The definition of MKR-PKE is outlined as follows.

Let $\mathcal{SK}$ denote the space of secret keys, which forms an abelian group under the operation $+$, and $\mathcal{PK}$ denote the space of public keys, which forms an abelian group under the operation $\cdot$. $\mathcal{M}$ denotes the plaintext space, which forms a group under the operation $+$, and $\mathcal{C}$ denotes the space of ciphertexts. MKR-PKE is a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Dec}, \mathsf{ReRand})$ s.t.:

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

MKR-PKE is an IND-CPA secure PKE scheme with the following properties:

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

$$\mathsf{ParDec}(sk_1, \mathsf{Enc}(pk_1 \cdot pk_2, x)) \overset{s}{\approx} \mathsf{Enc}(pk_2, x).$$

Additively homomorphic. There is an efficient operation $\boxplus$: $\mathcal{C} \times \mathcal{C} \to \mathcal{C}$ s.t. for any $pk \in \mathcal{PK}$ and any $x_1, x_2 \in \mathcal{M}$,

$$\mathsf{Enc}(pk, x_1) \boxplus \mathsf{Enc}(pk, x_2) \overset{s}{\approx} \mathsf{Enc}(pk, x_1 + x_2).$$

Additive homomorphism implies the following rerandomizable property:
Rerandomizable. For any $pk \in \mathcal{PK}$ and any $x \in \mathcal{M}$,

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

Beyond the above MKR-PKE definition, our framework additionally requires an efficient zero-preserving algorithm $\mathsf{ZeroRand}$, which takes as input a public key $pk \in \mathcal{PK}$ and a ciphertext $\mathsf{ct} \in \mathcal{C}$, outputs another ciphertext $\mathsf{ct}' \in \mathcal{C}$, s.t. the following zero-preserving randomizable property holds:
Zero-Preserving Randomizable. For any $pk \in \mathcal{PK}$ and any $x_1 \in \mathcal{M}$,

$$\mathsf{ZeroRand}(pk, \mathsf{Enc}(pk, x_1)) \overset{s}{\approx} \mathsf{Enc}(pk, x_2),$$

where if $x_1 = 0$, $x_2 = 0$; otherwise, x_2 is uniformly sampled from $\mathcal{M}$ using an independent randomness. This property ensures that rerandomization preserves encrypted zero values while transforming non-zero

encryptions into encrypted random values. We show that this property can be realized using homomorphic operations supported by additively homomorphic encryption, as long as the plaintext space $\mathcal{M}$ satisfies certain algebraic structures.

Additive homomorphism implies scalar multiplicative homomorphism, namely, there is an efficient operation $\boxdot$: $\mathbb{Z} \times \mathcal{C} \to \mathcal{C}$ s.t. for any $pk \in \mathcal{PK}$ and any $a \in \mathbb{Z}, x \in \mathcal{M}$,

$$a \boxdot \mathsf{Enc}(pk, x) \overset{s}{\approx} \mathsf{Enc}(pk, a \cdot x),$$

where $\cdot$ denotes scalar multiplication in group $\mathcal{M}$. A natural attempt is to define the zero-preserving rerandomization algorithm as:

$$\mathsf{ZeroRand}(pk, \mathsf{Enc}(pk, x_1)) = a \boxdot \mathsf{Enc}(pk, x_1),$$

where a is a uniformly sampled integer from $\mathbb{Z}_p$ (p is the order of group $\mathcal{M}$), using an independent randomness. By the scalar multiplicative homomorphism, we have $x_2 = a \cdot x_1$. It is easy to see that $x_2 = 0$ when $x_1 = 0$. However, to ensure that for any $x_1 \neq 0 \in \mathcal{M}$, the distribution of $a \cdot x_1$ is uniform over $\mathcal{M}$, every non-zero elements of group $\mathcal{M}$ must be generators. Only groups of prime order satisfy this condition. Therefore, EC MKR-PKE with plaintext space over EC groups of prime order (e.g., `NIST P-256` or `secp256k1`) is suitable for our work.

Another feasible instantiation leverages the fact that many additively homomorphic encryption schemes support the multiplication of ciphertexts by known plaintexts, i.e., there is an efficient operation $\boxdot$: $\mathcal{M} \times \mathcal{C} \to \mathcal{C}$ s.t. for any $pk \in \mathcal{PK}$ and any $x_1, x_2 \in \mathcal{M}$,

$$x_1 \boxdot \mathsf{Enc}(pk, x_2) \overset{s}{\approx} \mathsf{Enc}(pk, x_1 \cdot x_2),$$

where $\mathcal{M}$ forms a ring and $\cdot$ is the multiplication over ring $\mathcal{M}$. Now define the zero-preserving rerandomization algorithm as:

$$\mathsf{ZeroRand}(pk, \mathsf{Enc}(pk, x_1)) = b \boxdot \mathsf{Enc}(pk, x_1),$$

where b is a uniformly sampled elements from $\mathcal{M}$ using an independent randomness. We have $x_2 = b \cdot x_1$. It is easy to see that $x_2 = 0$ when $x_1 = 0$. However, to ensure that for any $x_1 \neq 0 \in \mathcal{M}$, the distribution of $a \cdot x_1$ is uniform over $\mathcal{M}$, every non-zero elements of ring $\mathcal{M}$ must be invertible, i.e., $\mathcal{M}$ is a field.

This field-structured MKR-PKE can be instantiated with the BFV [8, 20] or BGV [9] encryption schemes in the multi-user setting, specifying the plaintext modulus as a prime to ensure that the plaintext space forms a field. Here the noise in the resulting ciphertexts might reveal information about b, but this can be mitigated by the drowning technique, which is to add random noise from an interval that is λ larger than the maximum noise of ciphertexts. One benefit of this BFV/BGV MKR-PKE is to overcome the input space limitation inherent to prior MPSU protocols [24, 17] based on EC MKR-PKE (their input space is limited to EC points), which significantly restricts their applicability in real-world use cases [17].

H.2 Predicative Zero-Encryption

There are m ($m \geq 2$) parties with private inputs in a predicative zero-encryption protocol, where one of them is the receiver. If the truth-value of the associated first-order predicate formula Q in terms of their inputs is true, the receiver obtains an encrypted 0; otherwise it obtains an encrypted uniformly random value. The underlying encryption scheme is a MKR-PKE with a plaintext space that forms a field (assuming all parties have run the key-generation algorithm and collaboratively generated a common publicly key pk). The formal definition of predicative zero-encryption functionality is given in Figure 23.

Relaxed Predicative Zero-Encryption. In analogy to relaxed predicative zero-sharing, we relax predicative zero-encryption via a similar definition. Specifically, the relaxed predicative zero-encryption realizes the same functionality as Figure 23, except that if $Q(\mathbf{x}) = 0$, the plaintext is uniform but not necessarily independent of the joint view of any $t \leq m - 1$ parties in real execution.

By leveraging the additive homomorphism of MKR-PKE, the relaxed predicative zero-encryption can be straightforwardly constructed from relaxed predicative zero-sharing associated with the same first-order

Parameters: m parties $P_1, \cdots P_m$ with inputs $\mathbf{x} = (x_1, \cdots, x_m)$, where P_1 is the receiver. A first-order predicate formula Q. A MKR-PKE scheme with a plaintext space of field $\mathbb{F}$ and a common public key pk.
Functionality: On input x_i from each P_i, if $Q(\mathbf{x}) = 1$, give $\mathsf{Enc}(pk, 0)$ to P_1. Otherwise, sample $s \leftarrow \mathbb{F}$ and give $\mathsf{Enc}(pk, s)$ to P_1.

Fig. 23. Ideal functionality for predicative zero-encryption $\mathcal{F}_{\mathsf{PZE}}^{Q}$

predicate formula Q as follows: Assume that all parties have obtained a secret-sharing $[r] = (r_1, \cdots, r_m)$ from a relaxed predicative zero-sharing protocol associated with Q. Let each P_i encrypts its secret shares r_i, and that the parties (except the receiver) send the ciphertexts to the receiver. The receiver outputs the sum of all ciphertexts, i.e., $\mathsf{Enc}(pk, r_1) \boxplus \cdots \boxplus \mathsf{Enc}(pk, r_m) \overset{s}{\approx} \mathsf{Enc}(pk, r_1 + \cdots + r_m) \overset{s}{\approx} \mathsf{Enc}(pk, r)$. It is evident that the plaintext of the output ciphertext in relaxed predicative zero-encryption satisfies the same condition as the reconstructed secret r of the output secret-sharing in relaxed predicative zero-sharing.

From Relaxed to Standard Predicative Zero-Encryption. Assume that the receiver has obtained a ciphertext $\mathsf{Enc}(pk, r)$ from a relaxed predicative zero-encryption protocol, with the goal to generate a new ciphertext $\mathsf{Enc}(pk, s)$ meeting the standard predicative zero-encryption definition. The transformation method is realized by leveraging the additive homomorphism and zero-preserving rerandomizable property of MKR-PKE: 1) The receiver broadcasts $\mathsf{Enc}(pk, r)$. 2) Each P_i runs the zero-preserving rerandomization algorithm by performing a ciphertext multiplication by a uniformly sampled value b_i, resulting in a new ciphertext $c_i = \mathsf{ZeroRand}(pk, \mathsf{Enc}(pk, r)) = b_i \boxdot \mathsf{Enc}(pk, r) \overset{s}{\approx} \mathsf{Enc}(pk, b_i \cdot r)$. 3) P_i then rerandomizes c_i and sends it to the receiver. 4) The receiver outputs the sum of all ciphertexts, i.e., $c_1 \boxplus \cdots \boxplus c_m = \mathsf{Enc}(pk, b_1 \cdot r) \boxplus \cdots \boxplus \mathsf{Enc}(pk, b_m \cdot r) \overset{s}{\approx} \mathsf{Enc}(pk, (b_1 + \cdots b_m) \cdot r)$. As previous, we have $s = (b_1 + \cdots b_m) \cdot r$ in the end. The correctness and independence of the plaintext can be verified through the same analysis as in Section 4.3. The privacy is guaranteed by the IND-CPA security and rerandomizable property of MKR-PKE.

Remark 2. Given that the uniform plaintexts of the output ciphertexts in relaxed predicative zero-encryption depend on the joint view of some $t \leq m - 1$ parties, decryption of these ciphertexts after the predicative zero-encryption invocations may potentially leak information in the case of these t parties are in collusion. The transformation technique prevents this kind of leakage by eliminating the dependence. Looking ahead, it is the crucial step to ensure security against arbitrary collusion in our new MPSO framework.[12]

H.3 Membership Zero-Encryption

Membership zero-encryption is the encryption based counterpart of membership zero-sharing, where P_{pivot} is the party holding an element x (while each of the others holding a set) and the receiver. If the associated set predicate formula $Q(x, X_1, \cdots, X_{\mathsf{pivot}-1}, X_{\mathsf{pivot}+1}, \cdots, X_m) = 1$, P_{pivot} receives an encrypted 0, otherwise P_{pivot} receives an encrypted uniformly random value. A batched version of membership zero-encryption functionality is given in Figure 24.

In a similar manner to batch membership zero-sharing, we can define several variants of batch membership zero-encryption:

- Batch pure membership zero-encryption functionality (Figure 25);
- Batch pure non-membership zero-encryption functionality (Figure 26);
- Batch pure membership zero-encryption with payloads functionality (Figure 27).

The constructions of these functionalities follow the same route as discussed previously: First, the parties execute the corresponding variant of relaxed batch membership zero-sharing protocol for the given set

[12] For example, without this transformation step, our new MPSU protocol would be vulnerable to the same colluding attack as in [40], and thus inheriting the same non-collusion assumption.

Parameters: m parties $P_1, \cdots P_m$, where P_{pivot} is the only one holding n elements instead of n sets, and P_{pivot} is the receiver. A set membership predicate formula Q. Batch size n. A MKR-PKE scheme with a plaintext space of field $\mathbb{F}$ and a common public key pk.
Functionality: On input $\mathbf{x} = (x_1, \cdots, x_n)$ from P_{pivot} and $\mathbf{X}_j = (X_{j,1}, \cdots, X_{j,n})$ from each P_j ($j \in \{1, \cdots, m\} \setminus \{\mathsf{pivot}\}$), for $1 \leq i \leq n$, if $Q(x_i, X_{1,i}, X_{\mathsf{pivot}-1,i}, X_{\mathsf{pivot}+1,i}, \cdots, X_{m,i}) = 1$, give $\mathsf{Enc}(pk, 0)$ to P_{pivot}. Otherwise, sample $s_i \leftarrow \mathbb{F}$ and give $\mathsf{Enc}(pk, s_i)$ to P_{pivot}.

Fig. 24. Batch Membership Zero-Encryption Functionality $\mathcal{F}^{Q}_{\mathsf{bMZE}}$

Parameters: m parties $P_1, \cdots P_m$, where P_{pivot} is the only one holding n elements instead of n sets, and P_{pivot} is the receiver. Batch size n. A MKR-PKE scheme with a plaintext space of field $\mathbb{F}$ and a common public key pk.
Functionality: On input $\mathbf{x} = (x_1, \cdots, x_n)$ from P_{pivot} and $\mathbf{X}_j = (X_{j,1}, \cdots, X_{j,n})$ from each P_j ($j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$), for $1 \leq i \leq n$, if $\bigwedge_{j \in \{1, \cdots, m\} \setminus \{\mathsf{pivot}\}} (x_i \in X_{j,i}) = 1$, give $\mathsf{Enc}(pk, 0)$ to P_{pivot}. Otherwise, sample $s_i \leftarrow \mathbb{F}$ and give $\mathsf{Enc}(pk, s_i)$ to P_{pivot}.

Fig. 25. Batch Pure Membership Zero-Encryption Functionality $\mathcal{F}_{\mathsf{bpMZE}}$

Parameters: m parties $P_1, \cdots P_m$, where P_{pivot} is the only one holding n elements instead of n sets, and P_{pivot} is the receiver. Batch size n. A MKR-PKE scheme with a plaintext space of field $\mathbb{F}$ and a common public key pk.
Functionality: On input $\mathbf{x} = (x_1, \cdots, x_n)$ from P_{pivot} and $\mathbf{X}_j = (X_{j,1}, \cdots, X_{j,n})$ from each P_j ($j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$), for $1 \leq i \leq n$, if $\bigwedge_{j \in \{1, \cdots, m\} \setminus \{\mathsf{pivot}\}} x_i \notin X_{j,i} = 1$, give $\mathsf{Enc}(pk, 0)$ to P_{pivot}. Otherwise, sample $s_i \leftarrow \mathbb{F}$ and give $\mathsf{Enc}(pk, s_i)$ to P_{pivot}.

Fig. 26. Batch Pure Non-Membership Zero-Encryption Functionality $\mathcal{F}_{\mathsf{bpNMZE}}$

Parameters: m parties $P_1, \cdots P_m$, where P_{pivot} is the only one holding n elements instead of $2n$ sets, and P_{pivot} is the receiver. Batch size n. The mapping function $\mathsf{payload}_j()$ from P_j's elements to the associated payloads. A MKR-PKE scheme with a plaintext space of field $\mathbb{F}$ and a common public key pk.
Functionality: On input $\mathbf{x} = (x_1, \cdots, x_n)$ from P_{pivot}, $\mathbf{X}_j = (X_{j,1}, \cdots, X_{j,n})$ and $\mathbf{V}_j = (V_{j,1}, \cdots, V_{j,n})$ from each P_j ($j \in \{1, \cdots m\} \setminus \{\mathsf{pivot}\}$), for $1 \leq i \leq n$, if $\bigwedge_{j \in \{1, \cdots, m\} \setminus \{\mathsf{pivot}\}} (x_i \in X_{j,i}) = 1$, give $\mathsf{Enc}(pk, 0)$ and $\mathsf{Enc}(pk, \sum_{j \in \{1, \cdots, m\} \setminus \{\mathsf{pivot}\}} v_{j,i})$ to P_{pivot}, where $v_{j,i} = \mathsf{payload}_j(x_i) \in V_{j,i}$. Otherwise, sample $s_i \leftarrow \mathbb{F}$ and $w_i \leftarrow \mathbb{F}$, and give $\mathsf{Enc}(pk, s_i)$ and $\mathsf{Enc}(pk, w_i)$ to P_{pivot}.

Fig. 27. Batch Pure Membership Zero-Encryption with Payloads Functionality $\mathcal{F}_{\mathsf{bpMZEp}}$

predicate formula Q. Then, the receiver reconstructs the secrets in encrypted form to produce ciphertexts that satisfy the relaxed membership zero-encryption condition. Finally, the parties collaboratively transform these ciphertexts to meet the independence requirement of standard membership zero-encryption. Note that in the case of pure membership zero-encryption with payloads, the payload-encryption (i.e., the second ciphertext output to the receiver) does not need to satisfy the independence requirement. Therefore, the final transformation technique is only applied to the first output ciphertext.

H.4 Our MPSO Framework

We develop our new MPSO framework on top of our previous MPSO framework. In general, the first phase remains basically unchanged: for each subformula Q_i, the parties assign their input elements to bins. Instead of invoking batch membership zero-sharing, they invoke batch membership zero-encryption. As a result, for the element x in P_{pivot}'s each bin, if $x \in Y_i$ (where Y_i is the set represented by Q_i), P_{pivot} receives encrypted 0, otherwise encrypted random values.[13] The second phase was originally the invocation of multi-party secret-shared shuffle — which is to prevent information leakage through the order of the values — and the reconstruction. In this encryption based framework, it is replaced with its encryption based counterpart — collaborative decryption and shuffle procedure [24, 17], which enables P_1 to obtain all plaintexts in a randomly permuted order, and prevents any coalition of $m-1$ parties from knowing the permutation. This phase proceeds as follows: the parties first send their ciphertexts to P_1. Before sending ciphertexts, they may add the elements in the corresponding bins (appended with an all-zero string as previous) to the plaintexts, by leveraging the additive homomorphism of MKR-PKE. This element addition step still depends on the functionality. Then, P_1 rerandomizes the ciphertexts from the first phase, permutes them using a random permutation π_1, and sends these ciphertexts to P_2. P_2 performs a partial decryption on the received ciphertexts, rerandomizes them, permutes them using a random permutation π_2, and forwards these ciphertexts to P_3. This iterative process continues until the last party, P_m, returns its permuted partially decrypted ciphertexts to P_1. Finally, P_1 fully decrypts the ciphertexts and identifies all elements appended with all-zero strings to recover the desired output set (or outputs the number of zero as the cardinality).

Note that the circuit-MPSO functionalities cannot be realized within this encryption based framework. The complete MPSO and MPSO-card protocols are provided in Figure 31. Additionally, the optimization techniques described in Sections 6.2 and 6.3 remain applicable to the corresponding typical protocols. Formal descriptions of our encryption based MPSI/MPSI-card (Figure 28), MPSI-card-sum (Figure 29), and MPSU/MPSU-card (Figure 30) protocols are also included in this section.

Comparison with Our Previous MPSO Framework. Our secret-sharing based MPSO framework achieves the state-of-the-art asymptotic complexity for MPSI and its variants. However, the complexity of its MPSU instantiation, while optimal among symmetric-key based MPSU protocols, remains marginally inferior to the public-key based MPSU protocol in [17]. The latter achieves the state-of-the-art complexity for MPSU, where the computation and communication complexity of both leader and clients scale linearly with the number of parties m and the set size n. In contrast, our secret-sharing based framework incurs a quadratic leader complexity $O(m^2n)$, given that secret-sharing $O(mn)$ elements among m parties requires the leader to receive $O(mn)$ shares from each client and reconstruct $O(mn)$ secrets during the last step. By changing our framework to an encryption based version, our new MPSO framework reduces the leader's complexity to $O(sn)$ (where s is the number of subformulas in the CPF), which is not necessarily depends on m. Notably, when it is instantiated as MPSU, we have $s = m-1$, and the MPSU protocol have linear complexity for both leader and clients, matching the best known results from [17]. In conclusion, our encryption based MPSO framework achieves the state-of-the-art asymptotic complexity for all typical functionalities that have been studied before, including MPSI, MPSI-card, MPSI-card-sum and MPSU. However, this improvement in asymptotic complexity comes at the cost of practical efficiency, particularly in the online phase, since the encryption-based framework relies on public-key operations. In contrast, our secret-sharing based framework is built entirely on OT and symmetric-key operations, rendering it more efficient in practice.

[13] In the previous framework, these values were secret-shared among involved parties.

Parameters. m parties $P_1, \cdots, P_m$. Set size n. The element length l. Cuckoo hashing parameters: hash functions h_1, h_2, h_3 and number of bins B. A MKR-PKE scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Dec}, \mathsf{ReRand})$ with plaintext space of field $\mathbb{F}$.

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

1. **MKR-PKE key generation.** For $1 \leq i \leq m$, P_i runs $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$, and distributes its public key pk_i to other parties. Define $sk = sk_1 + \cdots + sk_m$ and all parties can compute the associated public key $pk = \prod_{i=1}^m pk_i$.
2. **Hashing to bin.** P_1 does $\mathcal{C}_1^1, \cdots, \mathcal{C}_1^B \leftarrow \mathsf{Cuckoo}^B_{h_1,h_2,h_3}(X_1)$. For $1 < j \leq m$, P_j does $\mathcal{T}_j^1, \cdots, \mathcal{T}_j^B \leftarrow \mathsf{Simple}^B_{h_1,h_2,h_3}(X_j)$.
3. **Batch pure membership zero-encryption.** All parties invoke $\mathcal{F}_{\mathsf{bpMZE}}$ of batch size B, where P_1 acts as P_{pivot} with inputs $\mathcal{C}_1^1, \cdots, \mathcal{C}_1^B$ and each P_j inputs $\mathcal{T}_j^1, \cdots, \mathcal{T}_j^B$ for $1 < j \leq m$. P_1 receives $\mathbf{e}_1 = (e_{1,1}, \cdots, e_{1,B})$.

MPSI.

4. **Collaborative decryption.** P_1 sends $\mathbf{e}_1$ to P_2. For $1 < j \leq m$, P_j defines $\mathbf{e}_j = (e_{j,1}, \cdots, e_{j,B})$ where $e_{j,b} = \mathsf{ParDec}(sk_j, e_{j-1,b})$ for $1 \leq b \leq B$. If $j \neq m$, P_j sends $\mathbf{e}_j$ to P_{j+1}, otherwise to P_1.
5. P_1 sets $Y = \emptyset$. For $1 \leq b \leq B$, P_1 computes $u_b = \mathsf{Dec}(sk_1, e_{m,b})$. If $u_i = 0$, P_1 adds the element in $\mathcal{C}_1^b$ to Y. Output Y.

MPSI-card.

4. **Collaborative decryption and shuffle.** P_1 sends $\mathbf{e}_1$ to P_2. For $1 < j \leq m$:
   (a) For $1 \leq b \leq B$, P_j computes $e'_{j,b} = \mathsf{ParDec}(sk_j, e_{j-1,b})$, $pk_{A_j} = pk_1 \cdot \prod_{d=j+1}^m pk_d$, and $e''_{j,b} = \mathsf{ReRand}(pk_{A_j}, e'_{j,b})$.
   (b) P_j samples $\pi_j : [B] \to [B]$. P_j defines $\mathbf{e}''_j = (e''_{j,1}, \cdots, e''_{j,B})$ and computes $\mathbf{e}_j = \pi_j(\mathbf{e}''_j)$. If $j \neq m$, P_j sends $\mathbf{e}_j$ to P_{j+1}, otherwise to P_1.
5. P_1 defines $\mathbf{u} = (u_1, \cdots, u_B)$. For $1 \leq b \leq B$, P_1 computes $u_b = \mathsf{Dec}(sk_1, e_{m,b})$. P_1 outputs $\mathsf{zero}(\mathbf{u})$.

Fig. 28. MPSI/MPSI-card

Parameters. Same as parameters in Figure 28.

Inputs. Each party P_i has input $X_i = \{x_i^1, \cdots, x_i^n\} \subseteq \{0,1\}^l$ and $V_i = \{v_i^1, \cdots, v_i^n\}$, with a mapping function $\mathsf{payload}_i()$ from elements to payloads.

1. **MKR-PKE key generation.** Same as step 1 in Figure 28.
2. **Hashing to bin.** Same as step 2 in Figure 28.
3. **Batch pure membership zero-encryption with payloads.** All parties invoke $\mathcal{F}_{\mathsf{bpMZEp}}$ of batch size B, where P_1 acts as P_{pivot} with inputs $\mathcal{C}_1^1, \cdots, \mathcal{C}_1^B$ and each of the remaining parties P_j inputs $(\mathcal{T}_j^1, \cdots, \mathcal{T}_j^B)$ and $(\mathcal{V}_j^1, \cdots, \mathcal{V}_j^B)$. P_1 receives $\mathbf{e}_1 = (e_{1,1}, \cdots, e_{1,B})$ and $\mathbf{w}_1 = (w_{1,1}, \cdots, w_{1,B})$.
4. For $1 \leq b \leq B$, if $\mathcal{C}_1^b$ is not an empty bin, P_1 sets $w_{1,b} = w_{1,b} + \mathsf{Enc}(pk, v)$, where v is the associated payload with the element in $\mathcal{C}_1^b$.
5. **Collaborative decryption and shuffle.** P_1 sends $\mathbf{e}_1$ and $\mathbf{w}_1$ to P_2. For $1 < j \leq m$:
   (a) For $1 \leq b \leq B$, P_j computes $e'_{j,b} = \mathsf{ParDec}(sk_j, e_{j-1,b})$, $pk_{A_j} = pk_1 \cdot \prod_{d=j+1}^m pk_d$, $e''_{j,b} = \mathsf{ReRand}(pk_{A_j}, e'_{j,b})$ and $w'_{j,b} = \mathsf{ReRand}(pk, w_{j-1,b})$.
   (b) P_j samples $\pi_j : [B] \to [B]$. P_j defines $\mathbf{e}''_j = (e''_{j,1}, \cdots, e''_{j,B})$ and $\mathbf{w}'_j = (w'_{j,1}, \cdots, w'_{j,B})$, then computes $\mathbf{e}_j = \pi_j(\mathbf{e}''_j)$ and $\mathbf{w}_j = \pi_j(\mathbf{w}'_j)$. If $j \neq m$, P_j sends $\mathbf{e}_j$ and $\mathbf{w}_j$ to P_{j+1}, otherwise to P_1.
6. P_1 defines $\mathbf{u} = (u_1, \cdots, u_B)$. For $1 \leq b \leq B$, P_1 computes $u_b = \mathsf{Dec}(sk_1, e_{m,b})$ and $s_1 = \sum_{1 \leq b \leq B \text{s.t.} u_b = 0} w_{m,b}$. P_1 outputs $\mathsf{zero}(\mathbf{u})$ and sends s_1 to P_2. For $1 < j \leq m$, P_j computes $s_j = \mathsf{ParDec}(sk_j, s_{j-1})$. If $j \neq m$, P_j sends s_j to P_{j+1}, otherwise to P_1. P_1 computes $s = \mathsf{Dec}(sk_1, s_m)$ and outputs s.

Fig. 29. MPSI-card-sum

Parameters. m parties $P_1, \cdots, P_m$. Set size n. The element length l. The all-zero string length l'. Cuckoo hashing parameters: hash functions h_1, h_2, h_3 and number of bins B. A MKR-PKE scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Dec}, \mathsf{ReRand})$ whose plaintext space is a field $\mathbb{F}$.

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

1. **MKR-PKE key generation.** For $1 \le i \le m$, P_i runs $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$, and distributes its public key pk_i to other parties. Define $sk = sk_1 + \cdots + sk_m$ and all parties can compute the associated public key $pk = \prod_{i=1}^m pk_i$.
2. **Hashing to bin.** P_1 does $\mathcal{T}_1^1, \cdots, \mathcal{T}_1^B \leftarrow \mathsf{Simple}^B_{h_1,h_2,h_3}(X_1)$. For $1 < j \le m$, P_j does $\mathcal{C}_j^1, \cdots, \mathcal{C}_j^B \leftarrow \mathsf{Cuckoo}^B_{h_1,h_2,h_3}(X_j)$ and $\mathcal{T}_j^1, \cdots, \mathcal{T}_j^B \leftarrow \mathsf{Simple}^B_{h_1,h_2,h_3}(X_j)$.
3. **Batch pure non-membership zero-encryption.** For $1 < j \le m$, $P_1, \cdots, P_j$ invoke $\mathcal{F}_{\mathsf{bpNMZE}}$ of batch size B, where P_j acts as P_{pivot} with inputs $\mathcal{C}_j^1, \cdots, \mathcal{C}_j^B$ and each $P_{j'}$ inputs $\mathcal{T}_{j'}^1, \cdots, \mathcal{T}_{j'}^B$ for $j' \in \{i_1, \cdots, i_q\} \setminus \{j\}$. P_j receives $\mathbf{e}_j = (e_{j,1}, \cdots, e_{j,B})$.

MPSU.

4. For $1 < j \le m$, $1 \le b \le B$, if $\mathcal{C}_j^b$ is not an empty bin, P_j computes $e'_{j,b} = e_{j,b} + \mathsf{Enc}(pk, x \| 0^{l'})$, where x is the element in $\mathcal{C}_j^b$, otherwise P_j samples $s'_{j,b} \leftarrow \{0,1\}^{l+l'}$ and computes $e'_{j,b} = \mathsf{Enc}(pk, s'_{j,b})$.
5. For $1 < j \le m$, P_j sends $\mathbf{e}'_j = (e'_{j,1}, \cdots, e'_{j,B})$ to P_1. P_1 defines $\mathbf{c}_1 = (c_{1,1}, \cdots, c_{1,(m-1)B})$, where $c_{1,(j-2)B+b} = e'_{j,b}$ for $1 < j \le m$, $1 \le b \le B$.
6. **Collaborative decryption and shuffle.** P_1 sends $\mathbf{c}_1$ to P_2. For $1 < j \le m$:
   (a) For $1 \le b \le (m-1)B$, P_j computes $c'_{j,b} = \mathsf{ParDec}(sk_j, c_{j-1,b})$, $pk_{A_j} = pk_1 \cdot \prod_{d=j+1}^m pk_d$, and $c''_{j,b} = \mathsf{ReRand}(pk_{A_j}, c'_{j,b})$.
   (b) P_j samples $\pi_j : [(m-1)B] \to [(m-1)B]$. P_j defines $\mathbf{c}''_j = (c''_{j,1}, \cdots, c''_{j,(m-1)B})$ and computes $\mathbf{c}_j = \pi_j(\mathbf{c}''_j)$. If $j \ne m$, P_j sends $\mathbf{c}_j$ to P_{j+1}, otherwise to P_1.
7. P_1 sets $Y = \emptyset$. For $1 \le b \le (m-1)B$, P_1 computes $u_b = \mathsf{Dec}(sk_1, c_{m,b})$. If $u_b = y \| 0^{l'}$ for some $y \in \{0,1\}^l$, P_1 update $Y = Y \cup \{y\}$. Output Y.

MPSU-card.

4. For $1 < j \le m$, P_j sends $\mathbf{e}_j$ to P_1. P_1 defines $\mathbf{c}_1 = (c_{1,1}, \cdots, c_{1,(m-1)B})$, where $c_{1,(j-2)B+b} = e_{j,b}$ for $1 < j \le m$, $1 \le b \le B$.
5. **Collaborative decryption and shuffle.** P_1 sends $\mathbf{c}_1$ to P_2. For $1 < j \le m$:
   (a) For $1 \le b \le (m-1)B$, P_j computes $c'_{j,b} = \mathsf{ParDec}(sk_j, c_{j-1,b})$, $pk_{A_j} = pk_1 \cdot \prod_{d=j+1}^m pk_d$, and $c''_{j,b} = \mathsf{ReRand}(pk_{A_j}, c'_{j,b})$.
   (b) P_j samples $\pi_j : [(m-1)B] \to [(m-1)B]$. P_j defines $\mathbf{c}''_j = (c''_{j,1}, \cdots, c''_{j,(m-1)B})$ and computes $\mathbf{c}_j = \pi_j(\mathbf{c}''_j)$. If $j \ne m$, P_j sends $\mathbf{c}_j$ to P_{j+1}, otherwise to P_1.
6. P_1 defines $\mathbf{u} = (u_1, \cdots, u_{(m-1)B})$. For $1 \le b \le (m-1)B$, P_1 computes $u_b = \mathsf{Dec}(sk_1, c_{m,b})$. P_1 outputs $n + \mathsf{zero}(\mathbf{u})$.

Fig. 30. MPSU/MPSU-card

Parameters. m parties $P_1, \cdots, P_m$. Set size n. The element length l. The all-zero string length l'. A constructible set Y represented as a CPF representation $\psi(x, X_1, \cdots, X_m)$. Cuckoo hashing parameters: hash functions h_1, h_2, h_3 and number of bins B. A MKR-PKE scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{ParDec}, \mathsf{Dec}, \mathsf{ReRand})$ whose plaintext space is a field $\mathbb{F}$.

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

1. **MKR-PKE key generation.** For $1 \le i \le m$, P_i runs $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\lambda)$, and distributes its public key pk_i to other parties. Define $sk = sk_1 + \cdots + sk_m$ and all parties can compute the associated public key $pk = \prod_{i=1}^m pk_i$.
2. **Hashing to bin.** For $1 \le i \le m$, P_i does $\mathcal{C}_i^1, \cdots, \mathcal{C}_i^B \leftarrow \mathsf{Cuckoo}_{h_1,h_2,h_3}^B(X_i)$ and $\mathcal{T}_i^1, \cdots, \mathcal{T}_i^B \leftarrow \mathsf{Simple}_{h_1,h_2,h_3}^B(X_i)$.
3. **Single subformula evaluation.** Let $\psi = Q_1 \vee \cdots \vee Q_s$. For the i-th subformula $Q_i(x, X_{i_1}, \cdots, X_{i_q})$ in ψ, where $1 \le i \le s, \{i_1, \cdots, i_q\} \subseteq \{1, \cdots, m\}$,
   (a) If $q = 1$, suppose $i_1 = \cdots = i_q = j$, then $Q_i(x, X_j) = (x \in X_j)$. For $1 \le b \le B$, if $\mathcal{C}_j^b$ is not an empty bin, P_j computes $e_{i,b} = \mathsf{Enc}(pk, 0)$, otherwise P_j chooses $s_{i,b}$ at random and computes $e_{i,b} = \mathsf{Enc}(pk, s_{i,b})$.
   (b) If $q > 1$, suppose Q_i is set-separable with respect to X_j for some $j \in \{i_1, \cdots, i_q\}$ and $Q_i(x, X_{i_1}, \cdots, X_{i_q}) = (x \in X_j) \wedge Q_i'(x, X_{i_1}, \cdots, X_{j-1}, X_{j+1}, \cdots, X_{i_q})$. The parties invoke $\mathcal{F}_{\mathsf{bMZE}}^{Q_i'}$ where P_j acts as P_{pivot} with inputs $\mathcal{C}_j^1, \cdots, \mathcal{C}_j^B$ and each $P_{j'}$ inputs $\mathcal{T}_{j'}^1, \cdots, \mathcal{T}_{j'}^B$ for $j' \in \{i_1, \cdots, i_q\} \setminus \{j\}$. P_j receives $\mathbf{e}_i = (e_{i,1}, \cdots, e_{i,B})$.

MPSO.

4. For $1 \le i \le s$, $1 \le b \le B$, if $\mathcal{C}_j^b$ is not an empty bin, P_j (the same j as step 3) computes $e'_{i,b} = e_{i,b} + \mathsf{Enc}(pk, x \| 0^{l'})$, where x is the element in $\mathcal{C}_j^b$, otherwise P_j samples $s'_{i,b} \leftarrow \{0,1\}^{l+l'}$ and computes $e'_{i,b} = \mathsf{Enc}(pk, s'_{i,b})$.
5. For $1 \le i \le s$, P_j (the same j as step 3) sends $\mathbf{e}'_i = (e'_{i,1}, \cdots, e'_{i,B})$ to P_1. P_1 defines $\mathbf{c}_1 = (c_{1,1}, \cdots, c_{1,sB})$, where $c_{1,(i-1)B+b} = e'_{i,b}$ for $1 \le i \le s$, $1 \le b \le B$.
6. **Collaborative decryption and shuffle.** P_1 sends $\mathbf{c}_1$ to P_2. For $1 < j \le m$:
   (a) For $1 \le i \le sB$, P_j computes $c'_{j,i} = \mathsf{ParDec}(sk_j, c_{j-1,i})$, $pk_{A_j} = pk_1 \cdot \prod_{d=j+1}^m pk_d$, and $c''_{j,i} = \mathsf{ReRand}(pk_{A_j}, c'_{j,i})$.
   (b) P_j samples $\pi_j : [sB] \to [sB]$. P_j defines $\mathbf{c}''_j = (c''_{j,1}, \cdots, c''_{j,sB})$ and computes $\mathbf{c}_j = \pi_j(\mathbf{c}''_j)$. If $j \ne m$, P_j sends $\mathbf{c}_j$ to P_{j+1}, otherwise to P_1.
7. P_1 sets $Y = \emptyset$. For $1 \le i \le sB$, P_1 computes $u_i = \mathsf{Dec}(sk_1, c_{m,i})$. If $u_i = y \| 0^{l'}$ for some $y \in \{0,1\}^l$, P_1 update $Y = Y \cup \{y\}$. Output Y.

MPSO-card.

4. For $1 \le i \le s$, P_j (the same j as step 3) sends $\mathbf{e}_i$ to P_1. P_1 defines $\mathbf{c}_1 = (c_{1,1}, \cdots, c_{1,sB})$, where $c_{1,(i-1)B+b} = e_{i,b}$ for $1 \le i \le s$, $1 \le b \le B$.
5. **Collaborative decryption and shuffle.** P_1 sends $\mathbf{c}_1$ to P_2. For $1 < j \le m$:
   (a) For $1 \le i \le sB$, P_j computes $c'_{j,i} = \mathsf{ParDec}(sk_j, c_{j-1,i})$, $pk_{A_j} = pk_1 \cdot \prod_{d=j+1}^m pk_d$, and $c''_{j,i} = \mathsf{ReRand}(pk_{A_j}, c'_{j,i})$.
   (b) P_j samples $\pi_j : [sB] \to [sB]$. P_j defines $\mathbf{c}''_j = (c''_{j,1}, \cdots, c''_{j,sB})$ and computes $\mathbf{c}_j = \pi_j(\mathbf{c}''_j)$. If $j \ne m$, P_j sends $\mathbf{c}_j$ to P_{j+1}, otherwise to P_1.
6. P_1 defines $\mathbf{u} = (u_1, \cdots, u_{sB})$. For $1 \le i \le sB$, P_1 computes $u_i = \mathsf{Dec}(sk_1, c_{m,i})$. P_1 outputs $\mathsf{zero}(\mathbf{u})$.

Fig. 31. MPSO/MPSO-card