Practical Multi-party Private Set Intersection Cardinality and Intersection-Sum Under Arbitrary Collusion

You Chen[1], Ning Ding[1(✉)], Dawu Gu[1(✉)], and Yang Bian[2]

[1] School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai 200240, China
{chenyou99,dingning,dwgu}@sjtu.edu.cn

[2] Fudata Technology, Room 1003, T7, Lane 100, Pingjiaqiao Road, Shanghai 200126, China
douheng@fudata.cn

Abstract. Private set intersection cardinality (PSI-CA) and private intersection-sum with cardinality (PSI-CA-sum) are two primitives that enable data owners to learn the intersection cardinality of their data set, with the difference that PSI-CA-sum additionally outputs the sum of the associated integer values of all the data that belongs to the intersection (i.e., intersection-sum). In this paper, we investigate the practical constructions of these two primitives, focusing on the multi-party setting. To our knowledge, all existing multi-party PSI-CA (MPSI-CA) protocols are either impractical or vulnerable to arbitrary collusion (i.e., the adversary can corrupt any proper subset of all parties), and as for multi-party PSI-CA-sum (MPSI-CA-sum), there is even no formalization for this notion at present, not to mention secure constructions for it.

So in this paper, we first propose the first MPSI-CA protocol that achieves simultaneous practicality and security against arbitrary collusion (in the semi-honest adversary model). We also conduct implementation to verify its practicality (while the previous results under arbitrary collusion only present theoretical analysis of performance, lacking real implementation). Numeric results show that it only takes 12.805 s to finish the online computation by shifting expensive operations to an offline phase, even in the dishonest majority setting with 15 parties each holding 2^{16} data. Among all parties, the cost of clients is especially lower compared to that of the known results, which is only 0.3 s in finishing their tasks.

Second, we formalize the notion of MPSI-CA-sum and give the first realization which admits simultaneous practicality and security against arbitrary collusion as well. The computational complexity of it is roughly double that of our MPSI-CA protocol.

Besides the main results, we introduce the notions and provide efficient constructions of two new building blocks: multi-party secret-shared shuffle and oblivious zero-sum check, which may be of independent interest.

The original version of this chapter was revised: this chapter contained errors on page 8, 9, 10 & 11 in chapter 9 which is indicated in our final book. The correction to this chapter is available at https://doi.org/10.1007/978-3-031-26553-2_27

Y. Deng and M. Yung (Eds.): Inscrypt 2022, LNCS 13837, pp. 169–191, 2023.
https://doi.org/10.1007/978-3-031-26553-2_9

Keywords: Multi-party PSI-CA · Multi-party PSI-CA-sum · Secure multiparty computation

1 Introduction

Motivation. Private set intersection cardinality (PSI-CA) is a cryptographic primitive that enables multiple parties to learn the intersection cardinality of their private data sets without leaking other information beyond the intersection cardinality. PSI-CA can be applied to real-world applications like measuring advertisement conversion rates [10] and so on. Despite its broad usage, nevertheless, PSI-CA is still not sufficient for some applications where each data is associated with an integer value (e.g. payload), like measuring advertisement conversion rates when one person contributes multiple purchases [10]. Thus a variant of PSI-CA is proposed, known as private intersection-sum with cardinality (PSI-CA-sum) [10], which is specified to output the intersection cardinality, as well as the sum of associated payloads for all the elements that belong to the intersection (i.e., intersection-sum).

Besides measuring advertisement conversion rates, we come up with the following possible application of PSI-CA-sum. Consider a score-based voting scenario with multiple voters, where voter P_i can vote for any candidate $s \in \{0,1\}^*$ that he prefers, and the ballot of him is associated with a score for candidate s (s is the candidate's ID). If P_i does not vote for candidate s, then there is no need for him to give s a score. P_i's voting result is represented using a set $S_i = \{(s_{i,1}, v_i(s_{i,1}), ..., (s_{i,m}, v_i(s_{i,m})\}$ of size m, where $s_{i,k}, k \in [m]$ are the IDs of his chosen candidates and $v_i(s_{i,k})$ is his score of candidate $s_{i,k}$. Given the set $S_i, i \in [n]$ of n voters, the set of common candidates supported by all voters is denoted as set intersection IS. The total score of a common candidate s is $\sum_{i=1}^{n} v_i(s)$, which can be used to calculate the average score of every common candidate. In this problem setting, the required information consists of the intersection cardinality $|IS|$ and the sum of common candidates' scores Sum_{IS} (i.e., $Sum_{IS} = \sum_{i=1}^{n} \sum_{x \in IS} v_i(x)$), so that the average score of a common candidate is $Sum_{IS}/|IS|$. Here, PSI-CA-sum can be employed to securely obtain the average score without additional information leakage.

However, most existing PSI-CA protocols work in the two-party setting, while the results of multi-party PSI-CA (MPSI-CA) are either limited by massive computational overhead, or vulnerable to arbitrary collusion (i.e., the adversary can corrupt any proper subset of all parties [15]). Meanwhile, to the best of our knowledge, there has been no work for multi-party PSI-CA-sum (MPSI-CA-sum). Therefore, we will address the problems and aim at formalizing the notion of MPSI-CA-sum, proposing protocols for MPSI-CA and MPSI-CA-sum that can achieve simultaneous practicality and security against arbitrary collusion.

1.1 State of the Art of MPSI-CA

Although there have been some effective two-party PSI-CA schemes [5,7,13], only a small number of works can deal with the multi-party setting [1,2,11,17].

Existing constructions of PSI-CA protocols can be generally classified into three categories, depending on whether the protocol is based on circuits, public key operations, or oblivious transfer (OT) and its extensions, say oblivious programmable pseudorandom function (OPPRF). Previous MPSI-CA schemes secure against arbitrary collusion typically follow public-key-based paradigm, and their computational complexities are determined by the number of expensive public key operations. Kissner and Song [11] proposed the first MPSI-CA protocol in the semi-honest model. This protocol relies on polynomial evaluation and homomorphic encryption (HE), and the overall computational complexity of it is $O\left(n^2 m_{\max}^2\right)$, where n is the number of parties and m_{max} is the maximum set size. Debnath et al. [2] presented an MPSI-CA protocol based on inverse bloom filter (IBF) and HE. The protocols in [2,11] are both proven secure against arbitrary collusion. Despite their good properties in privacy preserving, it is impractical for resource-limited devices with large data sets to carry out these protocols due to the massive computational overhead.

To tackle with this problem, two practical schemes have been proposed. Chandran et al. [1] introduced a circuit-based generic multi-party computation protocol, which can be extended to realize MPSI-CA by modifying the circuit. However, this protocol is only proven secure with honest majority in semi-honest model. Besides, a concurrent work of [17] presented two OPPRF-based MPSI-CA protocols under the additional assumption that specific parties are non-colluding, which deviate from the well-known "threshold security". Although assuming the existence of some specific non-colluding parties can improve the performance, it is believed that the "threshold security" is closer to real life applications for the following reasons: (1) There may not always exist such well-established non-colluding parties to participate in the protocol; (2) The identities of corrupted parties may be kept secret to honest parties, so it is unrealistic to assume that specific parties are non-colluding and to appoint them to perform special tasks.

Therefore, how to design and implement a practical semi-honest secure MPSI-CA scheme under arbitrary collusion is still worth studying.

1.2 State of the Art of Two-Party PSI-CA-Sum

(Since there is no result of PSI-CA-sum in the multi-party setting) we sketch some known results on the two-party PSI-CA-sum [7,9,10]. Motivated by the business problem of online-to-offline advertisement conversions, Ion et al. [10] introduced the first two-party PSI-CA-sum protocol by applying the classic Diffie-Hellman style construction into this new scenario. The protocol then was further polished in [9] and developed into two new constructions, built on modern techniques like random OT, which nevertheless rely on expensive HE as a building block for aggregating intersection-sum. Garimella et al. [7] put forward a lightweight two-party PSI-CA protocol by adopting oblivious switching network and OT to successfully avoid the reliance on HE.

1.3 Our Contributions

In this paper we formalize the notion of MPSI-CA-sum and propose the first MPSI-CA protocol and MPSI-CA-sum protocol that can achieve simultaneous practicality and security against arbitrary collusion. Details are as follows.

MPSI-CA Under Arbitrary Collusion. Our MPSI-CA protocol admits the following properties and advantages.

- It is the first practical realization of MPSI-CA under arbitrary collusion to our knowledge, and we also conduct an implementation to verify its practicality (while the previous results under arbitrary collusion only present theoretical analysis of performance without real implementation).
- The cost of clients is especially lower than the existing schemes with the same security.
- Its computational efficiency is attributed to the element sharing technique and underlying lightweight primitives, which do not require any public key operations besides a set of base OTs.
- In our implementation, most of the expensive operations can be shifted to an offline phase to significantly reduce the running time of online computation. Numeric results show that even in the dishonest majority setting with 15 parties each holding 2^{16} data, it only takes 12.805 s to finish the online computation, which is about one fourth of the original running time.

Table 1 compares our MPSI-CA protocol with current MPSI-CA schemes with respect to security and computational complexity. On one hand, when compared to the existing practical schemes [1,17], our protocol is more secure, since the existing schemes are not resistant to arbitrary collusion (remark that our protocol is also of practicality which is incomparable to the schemes in [1,17] due to different running frameworks). On the other hand, when compared to the existing schemes secure against arbitrary collusion [2,11], our protocol is much more practical, since it adopts a set of base OTs and symmetric key operations to reduce the number of expensive public key operations.

Table 1. Comparison between MPSI-CA schemes

Comparison Between MPSI-CA Schemes		
MPSI-CA Schemes	Techniques	Security Model
[1]	OT+symmetric key operations	Honest majority
Server-aided [17]	OT+symmetric key operations	Two specific parties are non-colluding
Server-less [17]	OT+symmetric key operations	Three specific parties are non-colluding
[11]	HE	Arbitrary collusion
[2]	HE	Arbitrary collusion
Our Protocol 4.2	OT+symmetric key operations	Arbitrary collusion

Computational Complexities of MPSI-CA Schemes Under Arbitrary Collusion (Number of Public Key Operations)				
MPSI-CA Schemes	Primary Leader	Secondary Leader	Client	Total
[11]	/	/	$O(nm_{max}^2)$	$O(n^2m_{max}^2)$
[2]	$O(m_1)$	/	$O(km_{max})$	$O(knm_{max})$
Our Protocol 4.2	$O(t\kappa)$	$O(t\kappa)$	/	$O(t^2\kappa)$

MPSI-CA-sum Under Arbitrary Collusion. We formalize the notion of MPSI-CA-sum and propose the first MPSI-CA-sum protocol that achieves simultaneous practicality and security against arbitrary collusion. Its computational complexity is roughly double that of our MPSI-CA protocol. Compared with most two-party PSI-CA-sum schemes, our protocol avoids the usage of expensive HE in aggregating intersection-sum, thus greatly reducing the computational cost.

Additional Contributions. Besides the main contributions, we also introduce the new notions and efficient constructions of two new building blocks of our MPSI-CA and MPSI-CA-sum protocols: multi-party secret-shared shuffle and oblivious zero-sum check.

- Multi-party secret-shared shuffle helps multiple parties jointly shuffle the sum of their input data in an unknown permutation π and obtain additive secret shares of the result. It is an advancement of the multi-party Permute+Share [14] because it can hide π even when confronted with arbitrary collusion. Our construction is practical since its costly operations can be shifted to an offline phase.
- Oblivious zero-sum check is a primitive that can securely determine whether the sum of multiple parties' inputs is 0 without revealing anything else. Our construction of oblivious zero-sum check employs Beaver triples to reduce online computational overhead.

1.4 High-Level Description

In this part, we present a high-level overview of our MPSI-CA and MPSI-CA-sum protocols. Our protocols involve n parties, including $T = t+1$ leaders $L_1, ..., L_T$ and $n-T$ clients $P_1, ..., P_{n-T}$, where t is the corruption threshold (t can be up to $n-1$). In order to differentiate between leaders, leader L_1 is called the primary leader, and the rest of the leaders are called secondary leaders. Each party holds a private set with size m. The data set of the i-th leader L_i is $X_i, i \in [T]$, and that of the j-th client P_j is $S_j, j \in [n-T]$.

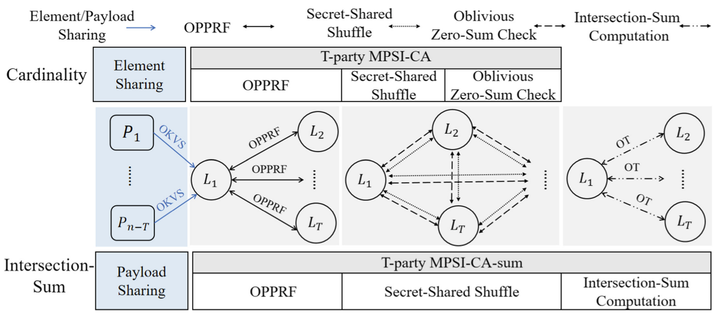

Fig. 1. The overview of our MPSI-CA and MPSI-CA-sum protocols

As shown in Fig. 1, in the setting of MPSI-CA, clients first share their encoded data sets to leaders through element sharing, so that the original n-party MPSI-CA problem can be reduced to T-party MPSI-CA of T leaders, where $T = t+1$.

Then, primary leader L_1 invokes OPPRFs with all secondary leaders $L_i, i \in [2, T]$ on each element $x_{1,k} \in X_1$. If $x_{1,k}$ belongs to the intersection, then the sum of all leaders' outputs and L_1's element sharing on $x_{1,k}$ equals 0, which is denoted as t_k. After participating in T-party secret-shared shuffle, each leader L_i obtains a random additive share of shuffled set $\{t_{\pi(k)}\}_{k\in[m]}$, where the shuffle order π is kept secret to all parties. Finally, leaders perform oblivious zero-sum check to securely calculate the number of elements that satisfy $\gamma_k t_{\pi(k)} = 0$, where the random value γ_k is unknown to any leader. If $\gamma_k t_{\pi(k)} = 0$, then L_1 adds one to intersection cardinality, otherwise the value of $t_{\pi(k)}$ will not be revealed.

In the setting of MPSI-CA-sum, parties need to perform element sharing (payload sharing), OPPRF and secret-shared shuffle on both elements and their associated payloads. After running oblivious zero-sum check on elements, L_1 can obtain a binary vector $\vec{e}$, which indicates the shuffled indices of elements that belong to the intersection. As for those elements, L_1 invokes OTs with all other leaders using choice string $\vec{e}$ to aggregate the sum of their associated payloads.

1.5 Organizations

Section 2 introduces the preliminaries. In Sect. 3, the notions and constructions of two new building blocks are presented. We propose the practical MPSI-CA and MPSI-CA-sum protocols in Sect. 4 and 5, respectively. The computational complexity of MPSI-CA-sum protocol is roughly double that of our MPSI-CA protocol, therefore we focus on implementing and analyzing the performance of our MPSI-CA protocol in Sect. 6.

2 Preliminaries

Notations. We use κ and λ to denote the computational and statistical security parameters. The set $\{1, 2, \dots, x\}$ is denoted as $[x]$ (thus $\sum_{i=1}^{T}$ is equivalent to $\sum_{i\in[T]}$). If the elements of a set $\{x_1, \dots, x_m\}$ are arranged in order, then this set can be expressed in the form of a vector $\vec{x} = (x_1, \dots, x_m)$. Therefore, $\vec{x}+\vec{y}$ means performing addition on corresponding elements in two sets x and y to obtain $\{x_1 + y_1, \dots, x_m + y_m\}$. Given a permutation π and a set $\vec{x} = (x_1, \dots, x_m)$, we represent the operation of shuffling the positions of elements in this set using permutation π with $\pi(\vec{x}) = (x_{\pi(1)}, \dots, x_{\pi(m)})$. The set intersection is denoted as IS, and the intersection cardinality is $|IS|$.

Security Definitions. The parties corrupted by a semi-honest adversary $\mathcal{A}$ will faithfully follow the protocol, while attempting to learn about other parties' inputs. Moreover, those corrupted parties will collude with each other. By "non-colluding parties", we mean that at most one of those parties can be corrupted by $\mathcal{A}$; while "arbitrary collusion" means that $\mathcal{A}$ may corrupt any proper subset of all parties, which is the most challenging case. The coalition of corrupted parties is denoted as $\mathcal{C}$. Let $\mathit{\Pi}$ be a protocol and f be a deterministic functionality.

We define the following distributions of random variables and use the real-ideal simulation paradigm to formally define the semi-honest security of Π [6]. In this paper, we prove the security of all the protocols based on Definition 1.

- $\mathrm{Real}_{\Pi}(\kappa, \mathcal{C}; x_1, \ldots, x_n)$: Each party P_i runs the protocol honestly using private input x_i and security parameter κ. Output $\{V_i \mid i \in \mathcal{C}\}, (y_1, \ldots, y_n)$, where V_i and y_i denote the final view and output of party P_i.
- $\mathrm{Ideal}_{f,\mathcal{S}}(\kappa, \mathcal{C}; x_1, \ldots, x_n)$: Compute $(y_1, \ldots, y_n) \leftarrow f(x_1, \ldots, x_n)$. Output $\mathcal{S}(\mathcal{C}, \{(x_i, y_i) \mid i \in \mathcal{C}\}), (y_1, \ldots, y_n)$, where $\mathcal{S}$ is a probabilistic polynomial time (PPT) simulator.

Definition 1. *[6] We say that protocol Π securely computes f in the presence of a semi-honest adversary, if there exists a PPT simulator $\mathcal{S}$ such that for $\mathcal{C}$ and all inputs $x_1, \ldots, x_n$, the distributions* $\mathrm{Real}_{\Pi}(\kappa, \mathcal{C}; x_1, \ldots, x_n)$ *and* $\mathrm{Ideal}_{f,\mathcal{S}}(\kappa, \mathcal{C}; x_1, \ldots, x_n)$ *are computationally indistinguishable in κ.*

Oblivious Key-Value Store (OKVS). The definitions of key-value store (KVS) and OKVS were first given in [8]. An OKVS is a generalized data structure that stores the mapping from keys to their values, and it can be instantiated with polynomial, garbled bloom filter (GBF) [4] and so on.

Definition 2. *[8] A KVS is parameterized by a set $\mathcal{K}$ of keys and a set $\mathcal{V}$ of values, and consists of two algorithms: (1)* Encode *takes as input a set of (k_i, v_i) key-value pairs and outputs an object S (or, with statistically small probability, an error indicator $\perp$); (2)* Decode *takes as input the object S, a key k and outputs a value v. A KVS is correct if, for all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys:*
$(k, v) \in A$ *and* $\perp \neq S \leftarrow \mathrm{Encode}(A) \Longrightarrow \mathrm{Decode}(S, k) = v$
A KVS is an OKVS if, for any two sets $\mathcal{K}^0, \mathcal{K}^1$ of m distinct keys, the output of $\mathcal{R}(\mathcal{K}^1)$ is computationally indistinguishable to that of $\mathcal{R}(\mathcal{K}^0)$, where:

$\mathcal{R}(\mathcal{K} = (k_1, \ldots, k_m))$
1. For $i \in [m]$: choose uniform $v_i \leftarrow \mathcal{V}$; 2. Return $\mathrm{Encode}(\{(k_1, v_1), \ldots (k_m, v_m)\})$.

Oblivious Programmable Pseudorandom Function (OPPRF, $\mathcal{F}_{\mathbf{opprf}}^{F,m,u}$). The formal definition of OPPRF was first given in [12], which also provided a semi-honest secure realization. An OPPRF takes as input the queries $(q_1, \ldots, q_u)$ from receiver and a programmed set $\mathcal{P} = \{\langle x_i, y_i \rangle\}_{i \in [m]}$ from sender. Then, the receiver's OPPRF outputs satisfy the following property: if the query $q_j = x_i \in \mathcal{P}$, then its OPPRF output equals y_i, otherwise the output is pseudorandom. Generally speaking, receiver's OPPRF outputs are fixed at some selected points.

Functionality 1: OPPRF $\mathcal{F}_{\mathbf{opprf}}^{F,m,u}$
Parameters: A pseudorandom function (PRF) F; upper bound m on the number of points to be programmed, and bound u on the number of queries.
Behaviour: On input $\mathcal{P}$ from the sender and u queries $(q_1, \ldots, q_u)$ from the receiver, where $\mathcal{P} = \{\langle x_1, y_1\rangle, \ldots, \langle x_m, y_m\rangle\}$ is a set of points:
•Run KeyGen $((1^\kappa, \mathcal{P})) \to (k, hint)$ and give $(k, hint)$ to the sender, where k is the PRF key and $hint$ stores the information of the set $\mathcal{P}$.
•Give $(hint, F(k, hint, q_1), \ldots, F(k, hint, q_u))$ to the receiver.

Multi-party Pemute+Share ($\mathcal{F}_{\mathrm{mPS}}^{T,m,i}$). $\mathcal{F}_{\mathrm{mPS}}^{T,m,i}$ takes as input the vectors $\vec{x_j}$ from all parties $P_j, j \in [T]$ and a permutation π_i from sender P_i, then outputs additive shares of shuffled sum $\pi_i(\sum_{j=1}^{T} \vec{x}_j)$ to every party. The functionality of $\mathcal{F}_{\mathrm{mPS}}^{T,m,i}$ was given in [14], along with an realization of $\mathcal{F}_{\mathrm{mPS}}^{T,m,i}$ based on OT and switching network, which is proven secure against a semi-honest adversary which may corrupt up to $T-1$ parties. $\mathcal{F}_{\mathrm{mPS}}^{T,m,i}$ is an essential building block of our multi-party secret-shared shuffle primitive proposed in Sect. 3.

Functionality 2: Multi-party Pemute+Share $\mathcal{F}_{\mathbf{mPS}}^{T,m,i}$
Parameters: T parties $P_j, j \in [T]$; the dimension of vector is m; the sender is P_i.
Behaviour: On input permutation π_i and vector $\vec{x}_i = (x_{i,1}, \ldots, x_{i,m})$ from sender P_i, and input vector $\vec{x}_j = (x_{j,1}, \ldots, x_{j,m})$ from each receiver $P_j, j \in [T]\backslash\{i\}$:
•Give shuffled share $\vec{x}'_j = (x'_{j,1}, \ldots, x'_{j,m})$ to all parties $P_j, j \in [T]$, where

$$\sum_{j\in[T]} x'_{j,k} = \sum_{j\in[T]} x_{j,\pi_i(k)}, k \in [m], \text{namely} \sum_{j\in[T]} \vec{x}'_j = \pi_i(\sum_{j\in[T]} \vec{x}_j).$$

3 Two New Primitives and Constructions

In this section, we present the notions and constructions of two new building blocks for our MPSI-CA and MPSI-CA-sum protocols, namely the multi-party secret-shared shuffle and oblivious zero-sum check.

3.1 Multi-party Secret-Shared Shuffle

We formalize the new notion of multi-party secret-shared shuffle, and give a realization of it. It can help parties shuffle the sum of their inputs in an unknown permutation order π, and obtain additive shares of the result.

Functionality ($\mathcal{F}_{\mathrm{mSS}}^{T,m}$). $\mathcal{F}_{\mathrm{mSS}}^{T,m}$ can be regarded as an advancement of the original $\mathcal{F}_{\mathrm{mPS}}^{T,m,i}$, since it ensures that none of the parties gets to know the permutation π. $\mathcal{F}_{\mathrm{mSS}}^{T,m}$ receives permutations π_i and vectors $\vec{x_i}$ from all parties $P_i, i \in [T]$, and gives them the additive shares of shuffled sum of inputs $\pi(\sum_{i\in[T]} \vec{x}_i)$ as outputs.

Functionality 3: Multi-party Secret-Shared Shuffle $\mathcal{F}_{\text{mSS}}^{T,m}$
Parameters: T parties $P_i, i \in [T]$; the dimension of vector is m.
Behaviour: On input permutation π_i and vector $\vec{x}_i = (x_{i,1}, \ldots, x_{i,m})$ from all parties $P_i, i \in [T]$:
- Give each party $P_i, i \in [T]$ an additive share $\vec{x}'_i = (x'_{i,1}, \ldots, x'_{i,m})$, where $\sum_{i\in[T]} x'_{i,k} = \sum_{i\in[T]} x_{i,\pi(k)}, k \in [m]$, namely $\sum_{i\in[T]} \vec{x}'_i = \pi(\sum_{i\in[T]} \vec{x}_i)$. Here, permutation $\pi = \pi_T \circ \ldots \pi_2 \circ \pi_1$ is the composition of T permutations.

Protocol. We propose a protocol to realize $\mathcal{F}_{\text{mSS}}^{T,m}$ as follows. This protocol invokes T rounds of T-party Permute+Share [14] in an iterative way. During the i-th round, P_i acts as the sender who provides permutation π_i and vector $\vec{x}_i^{(i-1)}$, others act as receivers with vectors $\vec{x}_j^{(i-1)}, j \in [T]\backslash\{i\}$ (Here, $\vec{x}_j^{(0)} = \vec{x}_j$, $j \in [T]$). Then, P_j receives an output $\vec{x}_j'^{(i-1)}$, where $\sum_{j\in[T]} \vec{x}_j'^{(i-1)} = \pi_i(\sum_{j\in[T]} \vec{x}_j^{(i-1)})$, and treats $\vec{x}_j'^{(i-1)}$ as his input vector during the next round. Finally, each party P_j obtains an additive share $\vec{x}_j'^{(T-1)}$ of the shuffled sum $\pi(\sum_{j\in[T]} \vec{x}_j^{(0)})$ with permutation $\pi = \pi_T \circ \cdots \circ \pi_1$. If we adopt the Permute+Share scheme proposed in [14], then our realization of $\mathcal{F}_{\text{mSS}}^{T,m}$ requires $O(T(T-1)m \log m)$ OTs in total.

Correctness. By the definition of $\mathcal{F}_{\text{mPS}}^{T,m,i}$, the sum of all parties' outputs equals $\pi_T(\sum_{j\in[T]} \vec{x}_j^{(T-1)}) = \pi_T(\pi_{T-1}(\sum_{j\in[T]} \vec{x}_j^{(T-2)})) = \cdots = \pi(\sum_{j\in[T]} \vec{x}_j^{(0)})$.

Theorem 1. *This protocol securely computes $\mathcal{F}_{\text{mSS}}^{T,m}$ under a semi-honest adversary which may corrupt up to $T-1$ parties, if $\mathcal{F}_{\text{mPS}}^{T,m}$ is secure against semi-honest adversaries.*

Proof. The views of corrupted parties (i.e., $\mathcal{C}$) consist of their inputs and views during T invocations of $\mathcal{F}_{\text{mPS}}^{T,m}$. As for the first round, simulator $\mathcal{S}$ chooses random vectors as corrupted parties' outputs by the definition of $\mathcal{F}_{\text{mPS}}^{T,m}$, then treats them as inputs into the next round. By following the above strategies for each round of T-party Permuta+Share and leveraging the simulator of subroutine functionality $\mathcal{F}_{\text{mPS}}^{T,m}$ in turn, the view of $\mathcal{C}$ during $\mathcal{F}_{\text{mSS}}^{T,m}$ can be ideally simulated by $\mathcal{S}$.

3.2 Oblivious Zero-Sum Check

We present the notion and construction of the new primitive of oblivious zero-sum check. It can help parties securely determine whether the sum of their inputs is 0 without revealing anything else. It can be employed in the last step of MPSI-CA to obtain the intersection cardinality of shuffled data.

Functionality ($\boldsymbol{\mathcal{F}_{\text{OZK}}^{T,m}}$). $\mathcal{F}_{\text{OZK}}^{T,m}$ receives input additive shares $\langle\vec{x}\rangle_i, i \in [T]$ from all parties, then outputs a binary vector $\vec{e} = (e_1, \ldots, e_m)$ to P_1. If the k-th position of the sum of input vectors $\vec{x} = \sum_{i=1}^{T} \langle\vec{x}\rangle_i$ equals 0, then $e_k = 1$; otherwise $e_k = 0$ (i.e., $e_k = 1$ only when $x_k = 0$). That is to say, $\mathcal{F}_{\text{OZK}}^{T,m}$ ensures that P_1 can not get to know the value of x_k unless it is equal to 0.

Functionality 4: Oblivious Zero-Sum Check $\mathcal{F}_{\text{OZK}}^{T,m}$
Parameters: The number of parties is T; the dimension of input vector is m.
Behaviour: On input vector $\langle\vec{x}\rangle_i$ from $P_i, i \in [T]$, where $\sum_{i=1}^{T}\langle\vec{x}\rangle_i = \vec{x} = (x_1, \ldots, x_m)$:
- Give a binary vector $\vec{e} = (e_1, \ldots, e_m)$ to P_1, where $e_k = 1$ if the k-th position of $\vec{x}$ equals 0 (i.e., $x_k = 0$), otherwise $e_k = 0$.

Protocol. As presented in Protocol 3.2, $\mathcal{F}_{\text{OZK}}^{T,m}$ can be realized using secret sharing mechanism. Since each party holds an additive share of secret $\vec{x}$, parties can obtain their additive shares of the product $\vec{\gamma} \cdot \vec{x}$ using Beaver multiplication triples, where $\vec{\gamma}$ is a "negotiated" random value and notation $\cdot$ denotes component-wise multiplication of two vectors. $\vec{\gamma}$ is kept secret to everyone, since each party P_i only knows an additive share $\langle\vec{\gamma}\rangle_i$ of $\vec{\gamma}$. If $x_k = 0$, it is obvious that the k-th position of $\vec{\gamma} \cdot \vec{x}$ equals 0 (i.e., $\gamma_k x_k = 0$); if $x_k \neq 0$, P_1 can not infer anything about x_k from $\gamma_k x_k$ due to the random value γ_k.

Parties need to interact with each other in order to obtain their additive shares of the product $\vec{\gamma} \cdot \vec{x}$. We note that $\vec{\gamma} \cdot \vec{x} = \sum_{i,j\in[T]}\langle\vec{\gamma}\rangle_i\langle\vec{x}\rangle_j$ can be divided into $\sum_{i\in[T]}\langle\vec{\gamma}\rangle_i\langle\vec{x}\rangle_i$ and $(T^2 - T)/2$ components $\langle\vec{\gamma}\rangle_i\langle\vec{x}\rangle_j + \langle\vec{\gamma}\rangle_j\langle\vec{x}\rangle_i$, where $i < j \in [T]$. For each component $\langle\vec{\gamma}\rangle_i\langle\vec{x}\rangle_j + \langle\vec{\gamma}\rangle_j\langle\vec{x}\rangle_i$, it is feasible for P_i and P_j to securely obtain their additive shares $sh_0^{i,j}$ and $sh_1^{i,j}$ using Beaver triples by following Protocol 3.2. The online pairwise share-based multiplication will be greatly accelerated by consuming the Beaver triples, which have already been prepared in the setup stage. Finally, P_i sends the sum of $\langle\vec{\gamma}\rangle_i\langle\vec{x}\rangle_i$ and his $T-1$ shares of $\sum_{j\in[T]\setminus\{i\}}(\langle\vec{\gamma}\rangle_i\langle\vec{x}\rangle_j + \langle\vec{\gamma}\rangle_j\langle\vec{x}\rangle_i)$ to P_1. So that P_1 can reconstruct $\vec{\gamma} \cdot \vec{x}$. If the k-th position of $\vec{\gamma} \cdot \vec{x}$ equals 0, P_1 sets e_k to 1, otherwise $e_k = 0$.

Correctness. It can be verified that $sh_0^{i,j} + sh_1^{i,j} = \langle\vec{\gamma}\rangle_i\langle\vec{x}\rangle_j + \langle\vec{\gamma}\rangle_j\langle\vec{x}\rangle_i$ based on the property of Beaver triples. Therefore, the sum of all parties' shares equals $\sum_{i\in[T]}\langle\vec{\gamma}\rangle_i\langle\vec{x}\rangle_i + \sum_{1\leq i<j\leq T}(\langle\vec{\gamma}\rangle_i\langle\vec{x}\rangle_j + \langle\vec{\gamma}\rangle_j\langle\vec{x}\rangle_i) = \vec{\gamma} \cdot \vec{x}$.

Theorem 2. *Protocol 3.2 securely computes $\mathcal{F}_{\text{OZK}}^{T,m}$ under a semi-honest adversary which may corrupt up to $T-1$ parties.*

Proof. In the trivial case that $P_1 \notin \mathcal{C}$, the views of corrupted parties $\mathcal{C}$ can be simulated by substituting all shares with random vectors. If $P_1 \in \mathcal{C}$, for those positions where $e_k = 0$, all generated and received shares of randomized $\gamma_k x_k$ are indistinguishable from uniformly random values; for positions where $e_k = 1$, shares can be simulated by choosing random values that sum to zero.

Protocol 3.2: Oblivious Zero-Sum Check
Parameters: The number of parties is T; the dimension of input vector is m.
Initialization: For every two parties P_i and P_j, $i, j \in [T], i < j$, they prepare enough Beaver triples $\langle\vec{a}\rangle_0$,$\langle\vec{b}\rangle_0$,$\langle\vec{c}\rangle_0$ and $\langle\vec{a}\rangle_1$,$\langle\vec{b}\rangle_1$,$\langle\vec{c}\rangle_1$ for online share-based multiplication, where $\vec{c} = \vec{a} \cdot \vec{b}$, $\vec{c} = \langle\vec{c}\rangle_0 + \langle\vec{c}\rangle_1$, $\vec{a} = \langle\vec{a}\rangle_0 + \langle\vec{a}\rangle_1$ and $\vec{b} = \langle\vec{b}\rangle_0 + \langle\vec{b}\rangle_1$. Note that P_i holds $\langle\vec{a}\rangle_0$,$\langle\vec{b}\rangle_0$,$\langle\vec{c}\rangle_0$, and P_j holds $\langle\vec{a}\rangle_1$,$\langle\vec{b}\rangle_1$,$\langle\vec{c}\rangle_1$. $\vec{a}$ and $\vec{b}$ are kept secret to both parties.
Input: Additive share $\langle\vec{x}\rangle_i$ from party P_i, where $\vec{x} = (x_1, \ldots, x_m) = \sum_{i=1}^{T} \langle\vec{x}\rangle_i$.
Output: P_1 outputs a binary vector $\vec{e} = (e_1, \ldots, e_m)$: if $x_k = 0$, then $e_k = 1$, otherwise $e_k = 0$.
Protocol:

1 For $i \in [T]$, each party P_i randomizes his share $\langle\vec{x}\rangle_i$ as follows:
  (a) **(Negotiating Randomness)** P_i locally generates a random vector $\langle\vec{\gamma}\rangle_i$, so that the random vector $\vec{\gamma} = \sum_{i=1}^{T} \langle\vec{\gamma}\rangle_i$ is unknown to everyone.
  (b) **(Pairwise Multiplication)** P_i computes his additive share of $\vec{\gamma} \cdot \vec{x} = \sum_{u,l \in [T]} \langle\vec{\gamma}\rangle_u \langle\vec{x}\rangle_l$. For each component $\langle\vec{\gamma}\rangle_i \langle\vec{x}\rangle_j + \langle\vec{\gamma}\rangle_j \langle\vec{x}\rangle_i, j \in [T] \backslash \{i\}$, P_i needs to interact with P_j as follows:
  - P_i locally computes $\langle\vec{\alpha}\rangle_0 = \langle\vec{x}\rangle_i - \langle\vec{a}\rangle_0$ and $\langle\vec{\beta}\rangle_0 = \langle\vec{\gamma}\rangle_i - \langle\vec{b}\rangle_0$, then announces them to P_j; P_j also locally computes $\langle\vec{\alpha}\rangle_1 = \langle\vec{x}\rangle_j - \langle\vec{a}\rangle_1$ and $\langle\vec{\beta}\rangle_1 = \langle\vec{\gamma}\rangle_j - \langle\vec{b}\rangle_1$, then announces them to P_i.
  - P_i reconstructs $\vec{\alpha}$ and $\vec{\beta}$, computes his additive share of $\langle\vec{\gamma}\rangle_i \langle\vec{x}\rangle_j + \langle\vec{\gamma}\rangle_j \langle\vec{x}\rangle_i$ as $sh_0^{i,j} = \langle\vec{c}\rangle_0 + \vec{\alpha} \cdot \langle\vec{b}\rangle_0 + \vec{\beta} \cdot \langle\vec{a}\rangle_0 + \vec{\alpha} \cdot \vec{\beta} - \langle\vec{\gamma}\rangle_i \langle\vec{x}\rangle_i$. P_j also obtains his additive share of $\langle\vec{\gamma}\rangle_i \langle\vec{x}\rangle_j + \langle\vec{\gamma}\rangle_j \langle\vec{x}\rangle_i$ as $sh_1^{i,j} = \langle\vec{c}\rangle_1 + \vec{\alpha} \cdot \langle\vec{b}\rangle_1 + \vec{\beta} \cdot \langle\vec{a}\rangle_1 + \vec{\alpha} \cdot \vec{\beta} - \langle\vec{\gamma}\rangle_j \langle\vec{x}\rangle_j$, where $sh_0^{i,j} + sh_1^{i,j} = \langle\vec{\gamma}\rangle_i \langle\vec{x}\rangle_j + \langle\vec{\gamma}\rangle_j \langle\vec{x}\rangle_i$.

2 **(Reconstruction)** Each $P_i, i \in [2, T]$ computes the sum of $\langle\vec{\gamma}\rangle_i \langle\vec{x}\rangle_i$ and all his shares of $\sum_{j=1, j \neq i}^{T} (\langle\vec{\gamma}\rangle_i \langle\vec{x}\rangle_j + \langle\vec{\gamma}\rangle_j \langle\vec{x}\rangle_i)$ (obtained in step 1(a)), and then sends the result to P_1, so that P_1 can reconstruct $\vec{\gamma} \cdot \vec{x}$. If the k-th position of $\vec{\gamma} \cdot \vec{x}$ equals 0, P_1 sets e_k to 1, otherwise $e_k = 0$.

4 MPSI-CA Protocol Under Arbitrary Collusion

In this section, we recall the functionality of MPSI-CA and propose a semi-honest secure MPSI-CA protocol under arbitrary collusion. First, we introduce a technique called element sharing to reduce the original n-party MPSI-CA to T-party MPSI-CA of T leaders. Then, a detailed description of our MPSI-CA protocol is presented.

Functionality ($\mathcal{F}_{\mathbf{MPSI-CA}}$). MPSI-CA allows n parties with m items to learn the intersection cardinality of their private sets without revealing anything else.

Functionality 5: MPSI-CA $\mathcal{F}_{\mathbf{MPSI\text{-}CA}}$
Parameters: T leaders $L_1, \ldots, L_T$; $n - T$ clients $P_1, \ldots, P_{n-T}$; the set size is m.
Behaviour: On input data sets X_i from all leaders $L_i, i \in [T]$, and data sets S_j from all clients $P_j, j \in [n - T]$:
- Give leader L_1 the intersection cardinality $|IS| = |(\bigcap_{i=1}^{T} X_i) \cap (\bigcap_{j=1}^{n-T} S_j)|$.

High-Level Description. The fundamental idea of our MPSI-CA protocol is to let all clients share their PRF-encoded data sets to T leaders $L_i, i \in [T]$, and then delegate leaders to complete the task of T-party PSI-CA. T is set to be $t+1$, otherwise the T-party MPSI-CA computation will be vulnerable to collusion attack in the worst case that all leaders are corrupted. Then, L_1 invokes OPPRFs with all secondary leaders. After that, all leaders treat their modified outputs $\vec{t_i}, i \in [T]$ as inputs to the following multi-party secret-shared shuffle and oblivious zero-sum check, so that L_1 can obtain the intersection cardinality.

4.1 Element Sharing

Considering that the overhead of MPSI-CA protocol tends to increase with the number of parties, it is a natural idea to delegate only a small number of parties to engage in expensive interactive procedures by sharing other parties' PRF-encoded data sets to them in the first step. This trick was first adopted by [15] and is called element sharing for short in this paper.

Sub-protocol 4.1: Element Sharing in MPSI-CA
Parameters: The number of parties is n, number of leaders is T; set size is m.
Input: $X_i = \{x_{i,1}, \ldots, x_{i,m}\}$ from leader $L_i, i \in [T]$; $S_j = \{s_{j,1}, \ldots, s_{j,m}\}$ from client $P_j, j \in [n-T]$.
Protocol:

1. **(Client)** For client $P_j, j \in [n-T]$,
   (a) He sends a random PRF key $K_{j,i}$ to each secondary leader $L_i, i \in [2, T]$.
   (b) For each element $s_{j,k} \in S_j, k \in [m]$, P_j computes the PRF-encoded value of $s_{i,j}$ as $\sum_{i=2}^{T} PRF(K_{j,i}, s_{j,k})$. Then, P_j encodes key-value pairs $\{\langle s_{j,k}, \sum_{i=2}^{T} PRF(K_{j,i}, s_{j,k})\rangle\}_{k\in[m]}$ into an OKVS D_j and sends D_j to primary leader L_1.
2. **(Primary Leader)** For each element $x_{1,k} \in X_1, k \in [m]$, L_1 decodes all received $D_j, j \in [n-T]$ on $x_{1,k}$ to get $D_j(x_{1,k})$, and then obtains his element sharing of $x_{1,k}$ as $q_1(x_{1,k}) = -\sum_{j=1}^{n-T} D_j(x_{1,k})$.
3. **(Secondary Leader)** Each secondary leader $L_i, i \in [2, T]$ computes the PRF outputs of all his elements $x_{i,k} \in X_i, k \in [m]$ using $n-T$ received keys $K_{j,i}, j \in [n-T]$, then adds the $n-T$ PRF outputs of $x_{i,k}$ together to obtain his element sharing of $x_{i,k}$ as $q_i(x_{i,k}) = \sum_{j=1}^{n-T} PRF(K_{j,i}, x_{i,k})$.

The functionality $\mathcal{F}_{\text{ElemSh}}^{n,T,m}$ of element sharing is that: for an element x, if $x \in IS$, then each leader $L_i, i \in [T]$ holds a random additive share $q_i(x)$ of 0 corresponding to x. The detailed process is shown in Sub-protocol 4.1, its correctness is obvious because if $x \in IS$, then each PRF key $K_{i,j}$ is used twice by both client P_j and leader L_i on the same item x, so that the two PRF outputs cancel out each other and $\sum_{i=1}^{T} q_i(x) = 0$.

We show that Sub-protocol 4.1 can securely compute $\mathcal{F}_{\text{ElemSh}}^{n,T,m}$ under a semi-honest adversary which may corrupt up to t parties ($t < n$) by giving a sketch of how to simulate the views of corrupted parties in the ideal world. The ideal views of corrupted clients are easy to simulate since they receive no messages. For corrupted $L_i, i \in [2, T]$, his received PRF keys can be simulated using random values. For the corrupted L_1, the OKVS D_j (from an honest party P_j)

appears random to him, since all the values encoded in D_j are encrypted using P_j's $T-1$ PRF keys. Therefore, $\mathcal{S}$ can easily simulate the OKVS by generating an OKVS that encode m random key-value pairs, which is computationally indistinguishable from his real view by the obliviousness property of OKVS.

4.2 Detailed Description

Protocol 4.2: MPSI-CA Under Arbitrary Collusion
Parameters: The set size is m; the number of leaders is $T = t+1$; hash functions h_1, h_2, h_3; the number of bins is b.
Input: $X_i = \{x_{i,1}, \ldots, x_{i,m}\}$ from leader L_i; $S_j = \{s_{j,1}, \ldots, s_{j,m}\}$ from client P_j.
Protocol:

1. **(Element sharing)** Run Sub-protocol 4.1 ($\mathcal{F}^{n,T,m}_{\text{ElemSh}}$). For each element $x_{i,k} \in X_i$, leader L_i obtains his element sharing of $x_{i,k}$ as $q_i(x_{i,k})$.
2. **(T-party MPSI-CA)** Leaders $L_i, i \in [T]$ act as follows:
   (a) **(Bucketing)** L_1 does $Table_1 \leftarrow \text{CuckooHash}^b_{h_1,h_2,h_3}(X_1)$, $L_i, i \in [2,T]$ does $Table_i \leftarrow \text{SimpleHash}^b_{h_1,h_2,h_3}(X_i)$.
   (b) **(OPPRF)** L_1 invokes $\mathcal{F}^{F,3m,b}_{\text{opprf}}$ with every $L_i, i \in [2,T]$,
   - Sender L_i provides a programmed set $\mathcal{P} = \{\mathcal{P}_k\}_{k\in[b]}$, where subset $\mathcal{P}_k = \{\langle x, q_i(x) - t_{i,k}\rangle\}_{x \in Table_i[k]}$ stores key-value pairs for the k-th bin $Table_i[k]$, and $t_{i,k}$ is a random value.
   - Receiver L_1 provides b queries $\{Table_1[k]\}_{k\in[b]}$, and outputs $\vec{r_i} = (r_{i,1}, \ldots, r_{i,b})$, where $r_{i,k}$ is the OPPRF output on $Table_1[k]$.

   (c) For each bin $k \in [b]$, L_1 computes $t_{1,k} = q_1(Table_1[k]) + \sum_{i=2}^{T} r_{i,k}$.
   (d) **(T-party Shuffle)** All leaders $L_i, i \in [T]$ jointly invoke $\mathcal{F}^{T,b}_{\text{mSS}}$.
   - Each L_i inputs the vector $\vec{t_i} = (t_{i,1}, \ldots, t_{i,b})$ and a permutation π_i, then outputs an additive share $\vec{t_i'}$ of the shuffled sum $\pi(\vec{t})$ (i.e., $\sum_{i=1}^{T} \vec{t_i'} = \pi(\vec{t})$), where $\vec{t} = \sum_{i=1}^{T} \vec{t_i} = (t_1, \ldots, t_b)$ and $\pi = \pi_T \circ \cdots \circ \pi_1$.

   (e) **(OZK)** All leaders $L_i, i \in [T]$ engage in $\mathcal{F}^{T,b}_{\text{OZK}}$ to securely obtain the number of zeros in the b-dimensional vector $\sum_{i=1}^{T} \vec{t_i'}$.
   - Each leader $L_i, i \in [T]$ inputs his share $\vec{t_i'}$ (obtained in step 2(d)).
   - L_1 outputs a binary vector $\vec{e}$ indicating which positions of $\sum_{i=1}^{T} \vec{t_i'}$ equal 0. If the k-th position is 0, then $e_k = 1$, otherwise $e_k = 0$.

   L_1 outputs the number of 1s in $\vec{e}$ as the intersection cardinality $|IS|$.

As shown in Protocol 4.2, leader L_i utilizes the bucketing technique [12] to hash his elements into a hash table $Table_i$ with b bins using simple hashing (or cuckoo hashing when $i = 1$) with hash functions h_1, h_2, h_3. For cuckoo hash table $Table_1$, each element $x \in X_1$ will be inserted into only one bin, say $Table_1[h_u(x)] = x$ for some $u \in [3]$. Finally, each empty bin will be padded with a dummy element. As for the simple hash table $Table_i, i \in [2,T]$, each $x \in X_i$ will be inserted into three bins $Table_i[h_1(x)]$, $Table_i[h_2(x)]$ and $Table_i[h_3(x)]$. When the number of hash functions is 3, the stash size can be reduced to 0 by setting $b = 1.28m$ while achieving a hashing failure probability of 2^{-40} [16].

After invoking $\mathcal{F}^{F,3m,b}_{\text{opprf}}$ on b queries $Table_1[k], k \in [b]$, if $Table_1[k] \in IS$, then leaders hold additive shares of 0 (i.e., $t_k = \sum_{i\in[T]} t_{i,k} = 0$). In order to obtain the number of k that satisfies $t_k = 0$ without revealing the index k, leaders invoke $\mathcal{F}_{\text{mSS}}^{T,m}$ to obtain additive shares of shuffled $t_{\pi(k)}$, where π is unknown to anyone. Then they engage in $\mathcal{F}_{\text{OZK}}^{T,m}$ to securely aggregate and rerandomize the value of $t_{\pi(k)}$. By the definition of $\mathcal{F}_{\text{OZK}}^{T,m}$, the output $\gamma_k t_{\pi(k)}$ equals 0 only when $t_{\pi(k)} = 0$, therefore L_1 adds one to intersection cardinality $|IS|$.

Correctness. If element $Table_1[k] \in IS$, then from the property of element sharing and OPPRF, we have $\sum_{i \in [T]} q_i(Table_1[k]) = 0$ and $r_{i,k} = q_i(Table_1[k]) - t_{i,k}$, and thus $t_k = 0$. By the correctness of multi-party secret-shared shuffle and oblivious zero-sum check, L_1 successfully reconstructs $\gamma_k t_{\pi(k)} = 0$, and knows there exists one more element that belongs to IS. Otherwise, if $Table_1[k]$ does not belong to some X_i or S_j, then either the OPPRF output $r_{i,k}$ or the OKVS decode output $q_1(Table_1[k])$ is a random value. Therefore, the probability that there exists an element $Table_1[k] \notin IS$ s.t. $\gamma_k t_{\pi(k)} = 0$ is negligible.

Theorem 3. *Protocol 4.2 securely computes $\mathcal{F}_{\text{MPSI-CA}}$ under a semi-honest adversary which may corrupt up to t parties ($t < n$), if $\mathcal{F}_{\text{ElemSh}}^{n,T,m}$, $\mathcal{F}_{\text{opprf}}^{F,3m,b}$, $\mathcal{F}_{\text{mSS}}^{T,b}$ and $\mathcal{F}_{\text{OZK}}^{T,b}$ are secure against semi-honest adversaries.*

Proof. We divide the proof into three cases.
Case1: ($L_i \notin \mathcal{C}, i \in [T]$). In this trival case, the views of corrupted parties (i.e., $\mathcal{C}$) can be easily simulated since they receive no messages.
Case2: ($L_1 \notin \mathcal{C}$). In this case, $\mathcal{C}$ receives no final output. The views of corrupted clients can be simulated in a way similar to Case 1. As for those corrupted $L_i, i \in [2, T]$, simulator $\mathcal{S}$ first chooses a random key k to simulate L_i's output of $\mathcal{F}_{\text{opprf}}^{F,3m,b}$ since $\mathcal{C}$ only sees the senders' views. Then, by the definition of $\mathcal{F}_{\text{mSS}}^{T,b}$, $\mathcal{S}$ chooses a random vector $\vec{t_i'}$ as his output of $\mathcal{F}_{\text{mSS}}^{T,b}$, and leverages the simulators of subroutine functionalities $\mathcal{F}_{\text{ElemSh}}^{n,T,m}$, $\mathcal{F}_{\text{mSS}}^{T,b}$, $\mathcal{F}_{\text{opprf}}^{F,3m,b}$ and $\mathcal{F}_{\text{OZK}}^{T,b}$ to simulate the view of corrupted L_i. The view output by $\mathcal{S}$ is indistinguishable from $\mathcal{C}$'s real view, which is obtained by the underlying simulators' indistinguishability.
Case3: ($L_1 \in \mathcal{C}$). In this case, $\mathcal{C}$ receives $|IS|$ as final output. $\mathcal{S}$ can simulate $\mathcal{C}$'s view as follows. In step $2(b)$, it simulates L_1's OPPRF outputs $\vec{r_i}, i \in [2, T]$ using uniformly random values while ensuring that: if $L_i \in \mathcal{C}$, then $r_{i,k} = q_i(Table_1[k]) - t_{i,k}$ for every element $Table_1[k]$ that belongs to $X_1 \cap X_i$, otherwise $r_{i,k}$ and $t_{i,k}$ are independent; if $L_i \notin \mathcal{C}$, then L_1's $r_{i,k}$ is picked at random. In step $2(d)$, by the definition of $\mathcal{F}_{\text{mSS}}^{T,b}$, it simulates corrupted parties' outputs of $\mathcal{F}_{\text{mSS}}^{T,b}$ using uniformly random vectors. In step $2(e)$, it simulates L_1's output $\vec{e}$ of $\mathcal{F}_{\text{OZK}}^{T,b}$ by uniformly sampling a binary vector with $|IS|$ ones due to the uniformly distributed permutation adopted in $\mathcal{F}_{\text{mSS}}^{T,b}$. After that, $\mathcal{S}$ can leverage the simulators of subroutine functionalities $\mathcal{F}_{\text{ElemSh}}^{n,T,m}$, $\mathcal{F}_{\text{mSS}}^{T,b}$, $\mathcal{F}_{\text{opprf}}^{F,3m,b}$ and $\mathcal{F}_{\text{OZK}}^{T,b}$ to simulate the view of $\mathcal{C}$. The view output by $\mathcal{S}$ is indistinguishable from $\mathcal{C}$'s real view, which is obtained by the underlying simulators' indistinguishability.

5 MPSI-CA-Sum Protocol Under Arbitrary Collusion

In this section, we first introduce a technique called payload sharing to share the payloads of clients to leaders. Then we smoothly extend Protocol 4.2 to provide a practical MPSI-CA-sum protocol that is secure under arbitrary collusion.

Functionality ($\mathcal{F}_{\mathbf{MPSI-CA-sum}}$). To the best of our knowledge, we are the first to formalize the notion of MPSI-CA-sum. The functionality of MPSI-CA-sum is a generalization of the two-party PSI-CA-sum proposed in [10], with some modifications as to the number of parties that hold the payloads. The associated payload of element x is denoted as $v_i(x)$ at leader L_i's side and $w_j(x)$ at client P_j's side, respectively. The purpose of MPSI-CA-sum is to securely output the $|IS|$ and intersection-sum Sum_{IS}, which is shown in Functionality 6.

Functionality 6: MPSI-CA-sum $\mathcal{F}_{\text{MPSI-CA-sum}}$
Parameters: T leaders $L_1, \ldots, L_T$; $n-T$ clients $P_1, \ldots, P_{n-T}$; the set size is m.
Behaviour: On input data set $X_i = \{x_{i,1}, \ldots, x_{i,m}\}$ and payload set $V_i = \{v_i(x_{i,1}), ...v_i(x_{i,m})\}$ from leader $L_i, i \in [T]$; data set $S_j = \{s_{j,1}, ...s_{j,m}\}$ and payload set $W_j = \{w_j(s_{j,1}), ...w_j(s_{j,m})\}$ from client $P_j, j \in [n-T]$:
• Give output $(|IS|, Sum_{IS})$ to leader L_1, where the intersection cardinality is $|IS| = |(\bigcap_{i=1}^{T} X_i) \cap (\bigcap_{j=1}^{n-T} S_j)|$, and the intersection-sum is $Sum_{IS} = \sum_{i=1}^{T} \sum_{x \in IS} v_i(x) + \sum_{j=1}^{n-T} \sum_{x \in IS} w_j(x)$.

High-Level Description. The procedures of our MPSI-CA-sum protocol are similar to those of Protocol 4.2. Parties perform payload sharing, OPPRF and shuffle on their associated payloads of each element, and run Protocol 4.2 in parallel to obtain a binary vector $\vec{e}$, which shows the shuffled indices of elements that belong to IS. As for those shuffled elements that belong to IS, L_1 invokes OTs with all other leaders using choice string $\vec{e}$, in order to aggregate the sum of their associated payloads (i.e., intersection-sum).

5.1 Payload Sharing

Sub-protocol 5.1: Payload Sharing in MPSI-CA-sum
Input: Set $X_i = \{x_{i,1}, \ldots, x_{i,m}\}$ and payload $V_i = \{v_i(x_{i,1}), \ldots v_i(x_{i,m})\}$ of leader L_i; Set $S_j = \{s_{j,1}, \ldots, s_{j,m}\}$ and payload $W_j = \{w_j(s_{j,1}), \ldots w_j(s_{j,m})\}$ of client P_j.
Protocol:

1. **(Client)** For client $P_j, j \in [n-T]$,
   (a) He sends a random PRF key $K'_{j,i}$ to each leader $L_i, i \in [T]$.
   (b) For each element $s_{j,k} \in S_j, k \in [m]$,
      - P_j computes its random mask $\sum_{i=1}^{T} PRF(K'_{j,i}, s_{j,k})$. So his masked payload of $s_{j,k}$ is $\widehat{w}_j(s_{j,k}) = w_j(s_{j,k}) + \sum_{i=1}^{T} PRF(K'_{j,i}, s_{j,k})$.
      - P_j performs (T,T) additive secret sharing on $\widehat{w}_j(s_{j,k})$, where the i-th share is denoted as $\widehat{w}_j^{(i)}(s_{j,k})$ (i.e., $\sum_{i=1}^{T} \widehat{w}_j^{(i)}(s_{j,k}) = \widehat{w}_j(s_{j,k})$).

   (c) For $i \in [T]$, P_j encodes the i-th set of key-value pairs $\{\langle s_{j,k}, \widehat{w}_j^{(i)}(s_{j,k})\rangle\}_{k\in[m]}$ into the i-th OKVS $DW_j^{(i)}$, and sends it to L_i.
2. **(Leader)** For each element $x_{i,k} \in X_i, k \in [m]$, $L_i, i \in [T]$ decodes all received $DW_j^{(i)}, j \in [n-T]$ on $x_{i,k}$ to obtain $DW_j^{(i)}(x_{i,k})$, and computes PRF outputs using all $n-T$ received PRF keys to obtain his payload sharing of $x_{i,k}$ as $\widehat{v}_i(x_{i,k}) = v_i(x_{i,k}) - \sum_{j=1}^{n-T} PRF(K'_{j,i}, x_{i,k}) + \sum_{j=1}^{n-T} DW_j^{(i)}(x_{i,k})$.

Sub-protocol 5.1 presents the steps of payload sharing, which aims to share the payloads of clients to T leaders. The functionality $\mathcal{F}_{\text{PaySh}}^{n,T,m}$ of payload sharing is that: for an element x, if $x \in IS$, then each leader $L_i, i \in [T]$ holds an additive share $\widehat{v}_i(x)$ of the sum of payloads corresponding to x (i.e., $\sum_{i=1}^{T} v_i(x) + \sum_{j=1}^{n-T} w_j(x)$). The procedures of payload sharing are similar to those of element sharing. The correctness of Sub-protocol 5.1 relies on the correctness of (T,T) additive secret sharing scheme and the property that the PRF output of input x with a fixed PRF key $K'_{i,j}$ is deterministic.

We show that Sub-protocol 5.1 can securely compute $\mathcal{F}_{\text{PaySh}}^{n,T,m}$ under a semi-honest adversary which may corrupt up to t parties ($t < n$) by briefly simulating the view of $\mathcal{C}$. The views of corrupted clients can be easily simulated since they receive no messages from the others. For corrupted $L_i, i \in [T]$, his received PRF key and OKVS from an honest party can be simulated using a random value and an OKVS that encodes m random key-value pairs, which are computationally indistinguishable from the real view by the obliviousness property of OKVS.

5.2 Detailed Description

The MPSI-CA-sum protocol under arbitrary collusion is presented in Protocol 5.2. Step 3 and step 4 can be executed in parallel by concatenating each element with its associated payload to avoid the cost of repeatedly invoking $\mathcal{F}_{\text{opprf}}^{F,3m,b}$ and $\mathcal{F}_{\text{mSS}}^{T,b}$. But note that there is no need to perform $\mathcal{F}_{\text{OZK}}^{T,b}$ on additive shares of the shuffled sum of payloads $\pi(\vec{g})$ (i.e., $\vec{g'_i}$, $i \in [T]$). The hash table $Table_i, i \in [T]$ used in step 4 is generated in step 3 by following step $2(a)$ of Protocol 4.2.

After invoking $\mathcal{F}_{\text{OZK}}^{T,b}$ during step 3, leader L_1 outputs a vector $\vec{e} = (e_1, \ldots, e_b)$. If $e_k = 1$, it means that the element in the $\pi^{-1}(k)$-th bin of $Table_1$ belongs to the intersection IS. Although L_1 can not infer the original index of this element (i.e., $\pi^{-1}(k)$) from k, he knows the existence of such an element. Therefore, he can still aggregate its associated payloads by invoking b OTs with each secondary leader $L_i, i \in [2, T]$. In the k-th OT with L_i, L_1 acts as a receiver with choice bit e_k, L_i acts as a sender who provides two strings $(mk_{i,k}, mk_{i,k} + g'_{i,k})$, where the random masks $mk_{i,k}, i \in [2, T], k \in [b]$ satisfy $\sum_{i=2}^{T} \sum_{k=1}^{b} mk_{i,k} = 0$. Those masks can be generated through additive secret sharing within secondary leaders. First, each secondary leader $L_i, i \in [2, T]$ locally generates a random vector $\vec{mk}'_i = (mk'_{i,1}, \ldots, mk'_{i,b})$ that ensures $\sum_{k=1}^{b} mk'_{i,k} = 0$. Then, $L_i, i \in [2, T]$ performs $(T-1, T-1)$ additive secret sharing on vector $\vec{mk}_i'$, and sends $T-2$ shares to other secondary leaders. Finally, $L_i, i \in [2, T]$ sums all his received shares and his local share together to obtain the new random mask vector $\vec{mk}_i$.

Correctness. Since the correctness of $|IS|$ (obtained in step 3) has already been proven in Section 4.2, here we only prove the correctness of Sum_{IS}. If $e_k = 1$, then we have $Table_1[\pi^{-1}(k)] \in IS$ and $\sum_{i=1}^{T} g'_{i,k} = \sum_{i=1}^{T} \widehat{v}_i(Table_1[\pi^{-1}(k)])$. After invoking OTs with each secondary leader, L_1 adds the $b(T-1)$ OT outputs together to obtain $\sum_{i=2}^{T} \sum_{k=1}^{b} mk_{i,k} + \sum_{e_k=1,k\in[b]} \sum_{i=2}^{T} g'_{i,k} = \sum_{e_k=1,k\in[b]} \sum_{i=2}^{T} g'_{i,k}$. Therefore, we have $\sum_{e_k=1,k\in[b]} \sum_{i=2}^{T} g'_{i,k} + \sum_{e_k=1,k\in[b]} g'_{1,k} = \sum_{x\in IS} \sum_{i=1}^{T} \widehat{v}_i(x) = \sum_{x\in IS} \left(\sum_{i=1}^{T} v_i(x) + \sum_{j=1}^{n-T} w_j(x) \right) = Sum_{IS}$.

Theorem 4. *Protocol 5.2 securely computes $\mathcal{F}_{\text{MPSI-CA-sum}}$ under a semi-honest adversary which may corrupt up to t parties ($t < n$), if $\mathcal{F}_{\text{ElemSh}}^{n,T,m}$, $\mathcal{F}_{\text{PaySh}}^{n,T,m}$ $\mathcal{F}_{\text{opprf}}^{F,3m,b}$, $\mathcal{F}_{\text{mSS}}^{T,b}$ and $\mathcal{F}_{\text{OZK}}^{T,b}$ and OT are secure against semi-honest adversaries.*

Proof. (sketch) The view of $\mathcal{C}$ during step 1–4 can be simulated by following similar strategies given in Theorem 3 of Protocol 4.2. Given $|IS|$, L_1's input choice string $\vec{e}$ can be simulated with a uniform binary vector with $|IS|$ ones. Since $\vec{mk}_i$ is a random mask, $\mathcal{S}$ can simulate corrupted L_i's OT inputs $(mk_{i,k}, mk_{i,k} + g'_{i,k})$ using random values, and simulate corrupted L_1's OT outputs from honest parties using random values, while ensuring that all $b(T-1)$ OT outputs sum to $Sum_{IS} - \sum_{e_k=1,k\in b} g'_{1,k}$. Then, $\mathcal{C}$'s view can be simulated by leveraging the simulator of underlying OT.

Protocol 5.2: MPSI-CA-sum Under Arbitrary Collusion

Parameters: The set size is m; the number of leaders is $T = t + 1$; hash functions h_1, h_2, h_3; the number of bins is b.

Input: Set $X_i = \{x_{i,1}, \ldots, x_{i,m}\}$ and payload $V_i = \{v_i(x_{i,1}), ...v_i(x_{i,m})\}$ of leader L_i; Set $S_j = \{s_{j,1}, \ldots, s_{j,m}\}$ and payload $W_j = \{w_j(s_{j,1}), ...w_j(s_{j,m})\}$ of client P_j.

Protocol:

1-2 **(Element/Payload Sharing)** Run Sub-protocol 4.1 ($\mathcal{F}^{n,T,m}_{\text{ElemSh}}$) and Sub-protocol 5.1 ($\mathcal{F}^{n,T,m}_{\text{PaySh}}$) in parallel. For each element $x_{i,k} \in X_i$, L_i obtains his element sharing and payload sharing of $x_{i,k}$ as $q_i(x_{i,k})$ and $\widehat{v}_i(x_{i,k})$.

3 **(T-party PSI-CA)** Run step 2 of Protocol 4.2, then L_1 will obtain a binary vector $\vec{e}$, where the number of 1s in $\vec{e}$ equals the intersection cardinality $|IS|$.

4 **(T-party MPSI-CA-sum)**
   (a) In step 3, each leader L_i has already obtained his hash table $Table_i$, so there is no need to repeat the bucketing step here.
   (b) **(OPPRF)** L_1 invokes $\mathcal{F}^{F,3m,b}_{\text{opprf}}$ with every $L_i, i \in [2, T]$,
   - Sender L_i provides a programmed set $\mathcal{P} = \{\mathcal{P}_k\}_{k\in[b]}$, where subset $\mathcal{P}_k = \{\langle x, \widehat{v}_i(x) - g_{i,k}\rangle\}_{x\in Table_i[k]}$ stores key-value pairs for the k-th bin $Table_i[k]$, and $g_{i,k}$ is a random value, $k \in [b]$.
   - Receiver L_1 provides b queries $\{Table_1[k]\}_{k\in[b]}$, and outputs $\vec{p_i} = (p_{i,1}, \ldots, p_{i,b})$, where $p_{i,k}$ is the OPPRF output on $Table_1[k]$,$k \in [b]$.

   (c) For each bin $k \in [b]$, L_1 computes $g_{1,k} = \widehat{v}_1(Table_1[k]) + \sum_{i=2}^{T} p_{i,k}$.
   (d) **(T-party Shuffle)** All leaders $L_i, i \in [T]$ jointly invoke $\mathcal{F}^{T,b}_{\text{mSS}}$.
   - Each L_i inputs the permutation π_i (adopted in step 3) and a vector $\vec{g_i} = (g_{i,1}, \ldots, g_{i,b})$, then outputs an additive share $\vec{g_i'}$ of the shuffled sum $\pi(\vec{g}) = \pi\left(\sum_{i=1}^{T} \vec{g_i}\right)$, where $\sum_{i=1}^{T} \vec{g_i'} = \pi(\vec{g})$ and $\pi = \pi_T \circ \cdots \circ \pi_1$.

5 **(Intersection-sum Computation)**
   (a) L_1 locally computes $\sum_{e_k=1,k\in[b]} g'_{1,k}$.
   (b) $L_2, ..., L_T$ jointly generate $T - 1$ random mask vectors $\vec{mk_i} = (mk_{i,1}, \ldots, mk_{i,b}), i \in [2, T]$, which satisfy that $\sum_{i=2}^{T} \sum_{k=1}^{b} mk_{i,k} = 0$.
   (c) For $i \in [2, T]$, L_1 invokes OTs with each secondary leader L_i.
   - Sender L_i inputs a set of strings $\{(mk_{i,k}, mk_{i,k} + g'_{i,k})\}_{k\in[b]}$.
   - Receiver L_1 inputs the choice string $\vec{e}$ and obtains b outputs. If $e_k = 0$, then the k-th output is $mk_{i,k}$, otherwise the k-th output is $mk_{i,k} + g'_{i,k}$.

6 L_1 adds $\sum_{e_k=1,k\in[b]} g'_{1,k}$ and all $b(T-1)$ OT outputs (received in step 5) together to obtain the intersection-sum Sum_{IS}.

6 Experimental Evaluation

Since the operations of computing intersection-sum is similar to those of computing intersection cardinality, the computational complexity of our MPSI-CA-sum protocol is roughly double that of our MPSI-CA protocol. So in this section, we only focus on evaluating the performance of our MPSI-CA protocol.

Parameters and Settings. We set statistical security parameter $\lambda = 40$ and computational security parameter $\kappa = 128$. We run our experiments on a laptop with an Intel i7-12700H 2.30 GHz CPU, 28 GB RAM, and Ubuntu-20.04 system in LAN setting. We instantiate the OPPRF using the realization provided in [12]. In the setup stage, it takes every two parties about 32 s to generate 2^{18} Beaver triples [3]. Each party adopts separated threads to communicate with others to ensure parallelism. Besides, we divide our protocol into offline and online phases in the experiment. The offline phase consists of all base OT operations in secret-shared shuffle, which can be carried out in advance because they are independent of the input sets. The online phase consists of all the remaining operations: element sharing, OPPRF, secret-shared shuffle (without base OT) and oblivious zero-sum check (without Beaver triples generation).

Running Time and Communication Cost of MPSI-CA (Protocol 4.2). Table 2 shows the running time of our MPSI-CA protocol in both online and offline phases, as well as its communication cost, which includes both sent and received messages.

We present the performance of clients and primary leader under three different corruption conditions, namely when $t = 1, n/2$ and $n - 1$. Assuming $t = 1$, it takes our MPSI-CA protocol only 27.174 s to compute the multi-party intersection cardinality of 15 parties, each with a large set size of 2^{18}. In the honest majority situation where $t = n/2$, the running time of leaders increases with the number of parties participating in multi-party secret-shared shuffle, and thus the running time is linear in t. When $n = 15$ and $m = 2^{12}$, the total running time is about 4.576 s with the online phase taking only 0.926 s. In the most challenging dishonest majority setting where $t = n - 1$, parties are not allowed to share their sets to leaders for fear of collusion attack, therefore, the number of leaders has to be n. However, since most of the expensive operations of multi-party secret-shared shuffle can be shifted to an offline phase, the total online running time can be reduced to only one fourth of the original time.

With respect to the communication performance of different parties, the cost of client is nearly independent of n and t. Whereas the cost of primary leader not only depends on n, but is also linear in the number of leaders $T = t + 1$. Concretely, when the set size is large (i.e., $m = 2^{18}$), our protocol takes roughly 7KB communication cost per item at each leader's side when $n = 5, t = 4$, which includes both sent and received messages. This cost increases to about 25 KB per item in the most challeging case that $n = 15$, $t = 14$ and $m = 2^{18}$.

Running Time of Different Steps in Protocol 4.2. Table 2 lists the running time of different steps in Protocol 4.2 when $t = n - 2$. As shown in the table, the two steps OPPRF and shuffle take a large percentage of the total running time. When n grows, the change in the running time of OPPRF is slight since each party adopts separated threads to communicate with others to ensure parallelism. In the case that $m = 2^{16}$, $n = 15$ and $t = 13$, it only takes 7.881 s to finish the online task of MPSI-CA computation with this simple optimization.

Table 2. The total running time, total communication cost, and the running time of different steps in our MPSI-CA protocol (Protocol 4.2).

n	t	Roles	Total Running Time (Seconds)				Total Communication Cost (MB)			
			$m = 2^{12}$	$m = 2^{14}$	$m = 2^{16}$	$m = 2^{18}$	$m = 2^{12}$	$m = 2^{14}$	$m = 2^{16}$	$m = 2^{18}$
5	1	Client	0.109	0.204	0.272	5.919	0.156265	0.625015	2.50002	10
		Leader	0.718	1.657	5.015	26.620	5.64464	25.4972	113.907	503.547
		Online	**0.467**	**1.354**	**4.433**	**24.110**				
	2	Client	0.110	0.221	2.77	5.921	0.156281	0.625031	2.50003	10
		Leader	1.022	2.339	6.867	36.352	10.6642	48.4943	217.814	967.094
		Online	**0.543**	**1.542**	**5.067**	**26.894**				
	4	Leader	2.650	4.025	12.165	52.312	20.7034	94.4886	425.628	1894.19
		Online	**0.670**	**1.789**	**6.289**	**31.236**				
10	1	Client	0.122	0.224	0.293	5.96	0.156265	0.625015	2.50002	10
		Leader	0.723	1.668	5.126	26.871	6.42595	28.6222	126.407	553.547
		Online	**0.497**	**1.360**	**4.567**	**24.516**				
	5	Client	0.111	2.37	0.309	5.994	0.156326	0.625076	2.50008	10.0001
		Leader	3.302	5.284	15.264	72.882	26.5044	120.611	542.036	2407.74
		Online	**0.844**	**1.890**	**5.733**	**32.764**				
	9	Leader	7.531	10.796	30.254	150.109	46.5828	212.599	957.664	4261.92
		Online	**1.004**	**2.690**	**10.415**	**58.485**				
15	1	Client	0.128	0.245	0.310	5.978	0.156265	0.625015	2.50002	10
		Leader	0.726	1.676	5.177	27.174	7.20725	31.7473	138.907	603.547
		Online	**0.505**	**1.383**	**4.598**	**25.115**				
	7	Client	0.111	0.239	0.349	6.183	0.156357	0.625107	2.50011	10.0001
		Leader	4.576	7.766	21.536	107.927	37.3249	169.73	762.35	3384.83
		Online	**0.926**	**2.257**	**5.955**	**36.913**				
	14	Leader	13.955	22.676	63.061	365.722	72.4621	313.97	1489.7	6629.66
		Online	**1.477**	**4.503**	**12.805**	**95.228**				

Running Time of Different Steps (Seconds)									
Steps	$n = 5, t = 3$			$n = 10, t = 8$			$n = 15, t = 13$		
	2^{12}	2^{14}	2^{16}	2^{12}	2^{14}	2^{16}	2^{12}	2^{14}	2^{16}
Element Sharing	0.119	0.250	0.319	0.109	0.237	0.341	0.103	0.263	0.337
OPPRF	0.473	1.261	5.113	0.566	1.407	4.697	0.942	1.821	5.691
Shuffle (Offline)	1.296	1.432	3.723	4.689	7.467	19.278	10.964	17.459	47.680
Shuffle (Online)	0.024	0.069	0.219	0.041	0.177	0.704	0.065	0.279	1.189
Oblivious Zero-Sum Check	0.006	0.009	0.017	0.013	0.021	0.036	0.019	0.034	0.065
Total	**1.938**	**3.062**	**9.560**	**5.485**	**9.425**	**25.368**	**12.269**	**20.115**	**55.561**
Online	**0.642**	**1.630**	**5.837**	**0.796**	**1.958**	**6.090**	**1.305**	**2.656**	**7.881**

Comparison with Other Works. There are only two MPSI-CA schemes [2,11] secure against arbitrary collusion in the semi-honest adversary model, but they only give theoretic analysis of performance without experimental results. Table 3 compares the performance of them and our MPSI-CA protocol (Protocol 4.2) in terms of computational and communication complexities.

As shown in Table 3, [2,11] both rely on a large number of expensive public key operations, which is linear in the maximal set size m_{max} or even m_{max}^2. Therefore, it is impractical for resource-limited devices with large data sets to carry out these protocols due to the massive computational overhead. Moreover, the efficiency of those schemes remains to be improved in the unbalanced data setting (i.e., the minimal set size $m_{min} \ll m_{max}$), or when the number of corrupted parties t only accounts for a small percentage of n.

By adopting lightweight primitives which do not require any public key operations besides a set of base OTs, the number of public key operations in our MPSI-CA protocols is independent of set size, which is significantly lower than that of [2,11]. At the same time, clients only need to send their PRF-encoded data to leaders instead of participating in expensive cryptographic interactive protocols for themselves, so that the total computational complexity can be significantly reduced especially when t/n is small. Besides, all the OTs required in multi-party secret-shared shuffle can be carried out in an offline phase, thus further decreasing the online computational complexity of our MPSI-CA protocol.

With respect to communication and round complexities, [2,11] both involve $O(n)$ rounds due to the operation of passing on randomized ciphertexts to the next party in a circle. Whereas we only need to perform T-party shuffle within $T = t+1$ leaders, and the round complexity is $O(t)$. Although utilizing expensive HE can save the communication cost during the stage of multi-party shuffle in [2], we reckon that the gap between [2] and our MPSI-CA scheme can be narrowed in an unbalanced setting, for we can designate the party with the smallest data set to be leader L_1 to ensure that $m_1 \ll m < m_{max}$. In this case, the additional communication overhead brought by multi-party secret-shared shuffle and oblivious zero-sum check can be reduced, so that the communication performance of our MPSI-CA scheme is comparable to that of [2].

Table 3. The computational and communication complexities of MPSI-CA schemes, where m is the average set size, m_{max} is the largest set size and m_1 is L_1's set size; k is the ratio of OKVS size to its encoded set size m. In our Protocol 4.2, most of the public key operations can be shifted to the offline phase.

Computational Complexity (Number of Public Key Operations)				
MPSI-CA Scheme	Primary Leader	Secondary Leader	Client	Total
[11]	/	/	$O(nm_{max}^2)$	$O(n^2 m_{max}^2)$
[2]	$O(m_1)$	/	$O(km_{max})$	$O(knm_{max})$
Our Protocol 4.2	$O(t\kappa)$	$O(t\kappa)$	/	$O(t^2\kappa)$
Computational Complexity (Number of Symmetric Key Operations)				
MPSI-CA Scheme	Primary Leader	Secondary Leader	Client	Total
Our Protocol 4.2	$O(tm_1 \log(m_1))$	$O((n-t)m + tm_1 \log(m_1))$	$O(tm)$	$O((n-t)tm + t^2 m_1 \log(m_1))$
Communication Complexity (Bits)				
MPSI-CA Scheme	Primary Leader	Secondary Leader	Client	Total
[11]	/	/	$O(nm_{max})$	$O(n^2 m_{max})$
[2]	$O(m_1)$	/	$O(km_{max})$	$O(knm_{max})$
Our Protocol 4.2	$O(tm_1 \log(m_1))$	$O(km + tm_1 \log(m_1))$	$O(km)$	$O(k(n-1)m + t^2 m_1 \log(m_1))$

Acknowledgement. We are very grateful to the reviewers for their valuable comments. This work was supported in part by the National Key Research and Development Project 2020YFA0712300.

References

1. Chandran, N., Dasgupta, N., Gupta, D., Obbattu, S.L.B., Sekar, S., Shah, A.: Efficient linear multiparty psi and extensions to circuit/quorum PSI. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 1182–1204 (2021)
2. Debnath, S.K., Stănică, P., Kundu, N., Choudhury, T.: Secure and efficient multiparty private set intersection cardinality. Adv. Math. Commun. **15**(2), 365 (2021)
3. Demmler, D., Schneider, T., Zohner, M.: Aby-a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
4. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol
5. Egert, R., Fischlin, M., Gens, D., Jacob, S., Senker, M., Tillmanns, J.: Privately computing set-union and set-intersection cardinality via bloom filters. In: Foo, E., Stebila, D. (eds.) ACISP 2015. LNCS, vol. 9144, pp. 413–430. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19962-7_24
6. Evans, D., Kolesnikov, V., Rosulek, M., et al.: A pragmatic introduction to secure multi-party computation. Found. Trends® Priv. Secur. **2**(2–3), 70–246 (2018)
7. Garimella, G., Mohassel, P., Rosulek, M., Sadeghian, S., Singh, J.: Private set operations from oblivious switching. In: Garay, J.A. (ed.) PKC 2021. LNCS, vol. 12711, pp. 591–617. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75248-4_21
8. Garimella, G., Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Oblivious key-value stores and amplification for private set intersection. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 395–425. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_14
9. Ion, M., et al.: On deploying secure computing: private intersection-sum-with-cardinality. In: 2020 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 370–389. IEEE (2020)
10. Ion, M., et al.: Private intersection-sum protocol with applications to attributing aggregate ad conversions. Cryptology ePrint Archive (2017)
11. Kissner, L., Song, D.: Private and threshold set-intersection. Carnegie-mellon univ pittsburgh pa dept of computer science. Technical report (2004)
12. Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: Practical multi-party private set intersection from symmetric-key techniques. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1257–1272 (2017)
13. Lv, S., et al.: Unbalanced private set intersection cardinality protocol with low communication cost. Futur. Gener. Comput. Syst. **102**, 1054–1061 (2020)
14. Mohassel, P., Sadeghian, S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 557–574. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_33

15. Nevo, O., Trieu, N., Yanai, A.: Simple, fast malicious multiparty private set intersection. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 1151–1165 (2021)
16. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. ACM Trans. Priv. Secur. (TOPS) **21**(2), 1–35 (2018)
17. Trieu, N., Yanai, A., Gao, J.: Multiparty private set intersection cardinality and its applications. Cryptology ePrint Archive (2022)