Practical Multi-party Private Set Intersection with Reducible Zero-sharing

Yewei Guan[†], Hua Guo[†§*], Man Ho Au[‡*], Jiarong Huo[†], Jin Tan[¶] and Zhenyu Guan[†]
[†]*School of Cyber Science and Technology, Beihang University, Beijing, China*
[§]*The State Key Laboratory of Blockchain and Data Security, Zhejiang University, Zhejiang, China*
[‡]*The Hong Kong Polytechnic University, Hong Kong, China*
[¶]*Independent Researcher*
Email:[†]{*ame_reiori, hguo, huojiarong, guanzhenyu*}*@buaa.edu.cn,*
[‡]*man-ho-allen.au@polyu.edu.hk,* [¶]*march1896@gmail.com*

***Abstract*—Multi-party Private Set Intersection (mPSI) enables $n(n \geq 3)$ parties, each holding a set of size m, to jointly compute their intersection while preserving the confidentiality of each set, which is essential for privacy-preserving data analysis and secure database queries. Existing mPSI protocols have limitations in achieving both sufficient security and practical efficiency.**

This paper presents a novel and efficient mPSI construction in the semi-honest model while resisting arbitrary collusion attacks. Our construction works in the offline/online paradigm. Given the corruption threshold t, the online phase achieves linear total computational and communication complexity, that is $O((n+t)m)$, and solely uses symmetric operations. This makes our construction theoretically outperform the existing works. The technical core of the construction is our newly extracted primitive called reducible zero-sharing, which allows $t(t < n)$ parties to obtain shares of zero for items in the intersection of n parties' input set, while resisting up to $t-1$ colluding parties. We present a practical construction of reducible zero-sharing in the offline/online paradigm by leveraging the homomorphic property of oblivious key-value store (OKVS).

With extensive experiments, we demonstrate that our construction outperforms state-of-the-art works in terms of online running time and communication cost. Specifically, compared to works with sufficient security, the online running time of our mPSI construction is $9.57 - 114.46\times$ faster in the LAN setting, $2.69 - 28.41\times$ faster in the WAN setting, while the communication cost is $0.29 - 28.70\times$ lower. Notably, the total performance (offline+online) still obtains up to $18.73\times$ improvement. Compared with works with practical efficiency, our mPSI construction achieves similar performance while providing stronger security.

1. Introduction

Two-party private set intersection (PSI) enables a sender and a receiver, each holding a private set, to learn the intersection of their sets without disclosing any additional information. Over the past decades, it has emerged as one of the most extensively studied primitives in secure multi-party computation [1], [2], [3], [4], [5]. Extending this concept, multi-party PSI (mPSI) generalizes two-party PSI to involve $n \geq 3$ parties. In recent years, mPSI has attracted considerable attention [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], leading to significant advancements in performance. mPSI sees a growing range of applications across various fields, such as federated learning [16], cooperative cache sharing [17], and network intrusion attack detection [8].

In contrast to two-party PSI, the primary challenge in designing mPSI in the semi-honest model lies in resisting collusion attacks. Such attacks allow any strict subset of parties to collude and potentially learn information beyond the intersection. Existing works provide two approaches to address the challenge.

Directly implementing mPSI [7], [14]: In such an approach, there are generally two phases, which we call the sharing phase and the reconstruction phase. In the sharing phase, each party maps every item in its set to a random share. Items in the intersection will be securely mapped to shares with specific property (e.g., the XOR-sum of the shares is 0). Then, in the reconstruction phase, one party collects shares and outputs the intersection with items whose corresponding shares hold the specific property. To resist arbitrary collusion under corruption threshold t, this approach requires $t+1$ parties to engage in a secure two-party protocol with all others in the sharing phase, using public-key operations. This leads to totally quadratic computational and communication complexity, that is $O(tn)$.

Reducing mPSI to PSI of fewer parties [11], [15]: Recently, a reduction paradigm [11] has been proposed, which reduces mPSI to PSI of fewer parties using only symmetric operations. With an efficient reduction phase, fewer parties engage in the expensive mPSI protocol. By leveraging the mPSI protocol with linear complexity, existing works following the reduction paradigm have achieved linear overall computational and communication complexity. However, they fail to achieve sufficient security. The mPSI instantiation in [11] is vulnerable to collusion attacks, illustrated in [14], while further work [15] based on the two-party PSI protocol relies on the non-collusion assumption

* *Corresponding Authors.*

that specific parties do not collude.

In summary, existing works have limitations in achieving both security and efficiency. Works that achieve sufficient security incur significant computational and communication costs, which are quadratic, while those with linear total complexity are vulnerable to collusion attacks. Given these limitations, we raise the following question:

In the semi-honest model, can we propose an mPSI construction with linear total computational and communication complexity, while resisting arbitrary collusion?

1.1. Technique Overview

We provide an answer to the aforementioned question by constructing an mPSI protocol with an offline phase, where parties are allowed to generate some input-independent correlated randomness. We focus on the mPSI task where there are n parties, each holding a set X_i of size m, and up to t colluding parties. The online phase of our construction achieves linear total computational and communication complexity while resisting arbitrary collusion. We now provide a high-level overview of our mPSI construction.

Design Insight. Our starting point is to address the challenge of achieving collusion resistance in mPSI. The common approach in existing works [7], [14] follows the zero-sharing+reconstruction paradigm. The core idea is to have parties first collectively and securely obtain a random share for each of their items in the zero-sharing phase, such that the XOR-sum of the obtained shares is zero for items in the intersection. In the subsequent reconstruction phase, one party queries the others for each of its items to receive the corresponding shares. If some item is in the intersection, the received shares XOR-sum to zero. We now briefly revisit the mechanism of this paradigm.

The zero-sharing phase can be realized by executing a two-party protocol $n-1$ times, where one party (e.g., P_1) acts as the sender and each of the remaining parties acts as a receiver. Specifically, for each item $x^1 \in X_1$, P_1 generates shares of zero $s^{1,j}$ such that $\bigoplus_{j=1}^{n} s^{1,j} = 0$. Then, P_1 engages in a two-party protocol with each party P_j, where P_j queries P_1 on the items in its set. For each item $x^j \in X_j$, if it also appears in X_1, then P_j obtains the corresponding share $s^{1,j}$; otherwise, P_j obtains a random share. Therefore, for items in the intersection, the XOR-sum of the obtained shares equals zero.

In the reconstruction phase, a designated Leader (e.g., P_n) securely retrieves the corresponding shares from the others for each item of its set and outputs the intersection. If P_n queries an item x^n that is also present in X_j, it receives the corresponding share $\hat{s}^j$ from P_j. Intuitively, if an item x^n is in the intersection, the XOR-sum of the received shares is zero; otherwise, it equals a random value.

While efficient in practice, this approach suffers from a key limitation that it resists collusion attacks only under the assumption that P_1 is honest. To achieve security against arbitrary collusions involving up to t parties, the first $t+1$ parties must act as the sender and interact with the others, ensuring that at least one party is honest. For every item in its set, each party XORs all obtained shares from the two-party protocol as its corresponding share. Therefore, this approach requires the overall $O(tn)$ interactions, resulting in significant computational and communication overhead.

Existing works [11], [15] have attempted to reduce the overhead by adopting the reduction paradigm. In this paradigm, $n-t-1$ parties play as clients, t parties play as servers, and the last party plays as a pivot. The clients first share their input sets to the possession of the pivot and t servers, while the servers and the pivot then engage in an expensive secure protocol to obtain the intersection. Inspired by this, we abstract it into a new primitive called *reducible zero-sharing*. Reducible zero-sharing is an n-party protocol, which takes the input set X_i of size m from each of n parties as inputs. After the protocol, there are only $t+1$ parties, called servers, each obtaining a mapping f_i such that for an item x in the intersection, the corresponding values $f_i(x)$ will XOR to zero. The remaining $n-t-1$ parties, called clients, output nothing. For security, we require a reducible zero-sharing $\mathcal{F}_{\mathsf{rzs}}^{n,t,m}$, which reduces n-party inputs to $t+1$ parties, to resist up to t colluding parties.

We observe that there are existing constructions that fit the definition of reducible zero-sharing. The instantiation in [11] first has clients share their input sets to the servers and the pivot, using symmetric operations. For each item in their input set, the servers and the pivot obtain shares of zero if it is also in the clients' input sets; otherwise, shares of a random value. Afterwards, to provide security, the servers and the pivot engage in an unconditional zero-sharing protocol [7] of $t+1$ parties. They utilize the result to mask the obtained shares. However, the unconditional zero-sharing protocol fails to provide security in the standard semi-honest model, making the instantiation vulnerable to collusion attacks, as analyzed in [14]. Directly replacing it with a secure zero-sharing protocol [7] will lead to significant overhead. Since each party is required to interact with each other as mentioned above, even with preprocessing techniques, it still introduces a total of $O(t^2)$ complexity.

Another approach is the bicentric zero-sharing [15] proposed by Gao et al. They let clients share their input sets to two specific parties with linear complexity. Their construction is solely based on symmetric operations, making it practically efficient. However, they rely on the non-collusion assumption that the two specific parties cannot collude.

Overall, existing instantiations fail to achieve both efficiency and security. To overcome this limitation, our goal is to design a practical and secure construction. Specifically, for efficiency, we aim for linear complexity and to rely solely on symmetric operations. In addition, for security, we aim to achieve sufficient security, which is resisting arbitrary collusions without any non-collusion assumption.

In this work, we propose an efficient reducible zero-sharing construction. We first describe a basic construction that achieves quadratic complexity and sufficient security. To optimize its performance, we introduce an offline phase, making it achieve both linear overall online complexity and sufficient security. With our reducible zero-sharing and a

reconstruction phase with linear complexity, we achieve a practical and secure mPSI protocol.

Our Construction of Reducible Zero-sharing. The main building block of our construction is the binary oblivious key-value store (binary OKVS) [5], [18]. A binary OKVS is a special case of OKVS, which consists of algorithms Encode and Decode. Encode encodes a set of key-value pairs $\{(k_i, v_i)\}$ to an OKVS object $\vec{S}$, represented as a vector. Decode allows querying the OKVS object $\vec{S}$ with any key k. Only when querying $k = k_i$, Decode outputs $v = v_i$. Note that both algorithms can be implemented using lightweight symmetric operations. Furthermore, a binary OKVS holds two homomorphic properties. The additive homomorphism is that if $\vec{S} = \vec{S}_1 \oplus \vec{S}_2$, then $\mathsf{Decode}(\vec{S}, k) = \mathsf{Decode}(\vec{S}_1, k) \oplus \mathsf{Decode}(\vec{S}_2, k)$. The scalar multiplicative homomorphism is derived from the additive homomorphism: if $\vec{S}' = b \cdot \vec{S}$, then $\mathsf{Decode}(\vec{S}', k) = b \cdot \mathsf{Decode}(\vec{S}, k)$.

Recall, the common approach to resisting collusion attack in the zero-sharing phase is to instantiate the two-party protocol using OPPRF [7], in which the sender learns a PRF key k and the receiver learns the output of PRF $F(k, r)$, as programmed by the sender, where F is a PRF and r is the receiver's input. However, since OPPRF relies on costly public-key operations, this approach is far from efficient. We observe that directly replacing OPPRF with OKVS can also achieve the functionality of zero-sharing, where the sender encodes an OKVS object with items as keys and shares as values, and the receiver can decode the object with its items to obtain the corresponding shares. However, this approach is vulnerable to collusion attacks due to the fundamental difference between OPPRF and OKVS. With OPPRF, parties obtain the same output on only a limited number of queries. In contrast, OKVS allows unlimited evaluations, and parties always obtain the same output when querying the same item, which enables the Leader to perform collusion attacks during the reconstruction phase (See [14] for details). To overcome this limitation, Wu et al. [14] proposed a hybrid approach. They combine OPRF and OKVS, where OPRF restricts the number of queries to ensure security and OKVS improves the efficiency. However, as OPRF heavily relies on public-key operations, the performance remains constrained.

In summary, resisting collusion attacks requires restricting the parties' ability to perform arbitrary queries. However, existing constructions based on O(P)PRF incur a total of $O(tn)$ interactions—each party interacting with all others—and involve numerous public-key operations, resulting in considerable overhead.

In this work, we propose a novel approach to restrict the querying capability of each party. Our key idea is to leverage the homomorphic property of binary OKVS. Specifically, suppose that each party P_i already holds an initial OKVS object $\vec{S}_i$ such that for arbitrary input x, the XOR-sum of $\mathsf{Decode}(\vec{S}_i, x)$ equals zero. Given a set X_i, party P_i decodes $\vec{S}_i$ with X_i and re-encodes an OKVS object $\vec{D}_i$ as

$$\vec{D}_i = \mathsf{Encode}(X_i, \mathsf{Decode}(\vec{S}_i, X_i)). \tag{1}$$

This ensures that only for an item $x \in \bigcap_i X_i$, the corresponding outputs of OKVS decoding XOR-sum to zero.

However, it remains vulnerable to collusion attacks as parties can query $\vec{S}_i$ on any item. Once colluding parties obtain all $\vec{D}_i$, they can distinguish whether an item $x \notin \bigcap_{i \in \mathsf{Corrupt}} X_i$ is held by all others by checking

$$\bigoplus_{i \in \mathsf{Corrupt}} \mathsf{Decode}(\vec{S}_i, x) = \bigoplus_{i \notin \mathsf{Corrupt}} \mathsf{Decode}(\vec{D}_i, x). \tag{2}$$

To address this challenge, we design a secure multi-party random scalar multiplication (mRSM) protocol to let parties mask the OKVS object $\vec{D}_i$. As a result, each party obtains $\vec{D}'_i$ such that

$$\bigoplus_i \vec{D}'_i = \Delta \cdot \bigoplus_i \vec{D}_i, \tag{3}$$

where Δ is a random scalar that no party learns. For any item $x \notin \bigcap_i X_i$, parties obtain shares of a random value using $\vec{D}'_i$ as there exists one OKVS object $\vec{D}_j$ such that $\mathsf{Decode}(\vec{D}_j, x)$ is a random value. In addition, parties cannot perform the collusion attack solely using $\vec{D}'_i$ and $\vec{S}_i$ due to the existence of Δ. Furthermore, for correctness, according to the scalar multiplication homomorphism of binary OKVS, for an item x in the intersection, the XOR-sum of $\mathsf{Decode}(\vec{D}'_i, x)$ remains zero, enabling the computation of the intersection.

The remaining challenge is how to make parties obtain the initial OKVS objects $\vec{S}_i$. In the context of reducible zero-sharing, we instantiate the output mapping with the OKVS object. Since only servers obtain the mapping, we let the clients share their input using OKVS, while the servers obtain the initial OKVS objects evaluated to shares of zero for items in the intersection of the clients' sets. Specifically, to resist up to t colluding parties, we consider the reducible zero-sharing $\mathcal{F}_{rzs}^{n,t,m}$. In the remainder of this paper, and without loss of generality, we refer to the first $n - t - 1$ parties as clients, the next $t + 1$ parties as servers, among which the last one, P_n, serves as the pivot. For each item $x_k \in X_i$, each client $P_i (i \in [1, n-t))$ samples $t+1$ random values $s_k^{i,j}$ for $j \in [n - t, n]$, such that $\bigoplus_{j=n-t}^{n} s_k^{i,j} = 0$. Then, each client P_i encodes $t + 1$ OKVS objects using all items x_k^i from X_i and the corresponding values $s_k^{i,j}$ as $\vec{S}_{ij} = \mathsf{Encode}(\{(x_k^i, s_k^{i,j})\})$, and sends $\vec{S}_{ij}$ to server P_j. After receiving $\vec{S}_{ij}$, P_j XORs these OKVS objects as $\vec{S}_j = \bigoplus_{i=1}^{n-t-1} \vec{S}_{ij}$.

According to the additive homomorphism, for an item x in the intersection of all clients' sets, each server P_j obtains the corresponding value y_j such that

$$y_j = \mathsf{Decode}(\vec{S}_j, x) = \bigoplus_{i=1}^{n-t-1} \mathsf{Decode}(\vec{S}_{ij}, x). \tag{4}$$

Note that $\mathsf{Decode}(\vec{S}_{ij}, x)$ for $j \in [n - t, n]$ are shares of zero, which are generated by P_i. Therefore, the XOR-sum

of all corresponding values y_j also equals zero.

$$\bigoplus_{j=n-t}^{n} y_j = \bigoplus_{j=n-t}^{n} \bigoplus_{i=1}^{n-t-1} \mathsf{Decode}(\vec{S}_{ij}, x) = 0. \quad (5)$$

For an item x outside the intersection of the clients' sets, there exists $\vec{S}_j$ such that $\mathsf{Decode}(\vec{S}_j, x)$ is random, thereby the XOR-sum of all corresponding values remains random.

Overall, our construction of reducible zero-sharing consists of two phases: a reduction phase and a mask phase. In the reduction phase, clients share their input sets to servers solely using OKVS, which is a lightweight cryptographic primitive. Then, in the mask phase, servers mask the obtained OKVS objects by engaging in an mRSM protocol.

Note that our basic construction incurs quadratic overall complexity. For example, in the reduction phase, each client sends an OKVS object to each server, resulting in a communication cost of $O(tn)$. To optimize our construction, we introduce preprocessing techniques. By offloading costly operations, we propose a reducible zero-sharing construction with linear total online complexity and sufficient security.

Generating Initial OKVS Objects with Seeds. We leverage the doubly obliviousness of binary OKVS, that is, if all keys and values are uniformly random, then the result OKVS object is statistically indistinguishable from a uniformly random vector. Therefore, the last OKVS object $\vec{S}_{i,n}$ can be generated by the remaining t random OKVS objects as

$$\vec{S}_{i,n} = \mathsf{Encode}(X_i, \mathsf{Decode}(\bigoplus_{j=n-t}^{n-1} \vec{S}_{ij}, X_i)), \quad (6)$$

where $\vec{S}_{ij}$ is generated by a PRG G as $\vec{S}_{ij} = G(s_{ij})$ and s_{ij} is a seed of length κ. Therefore, in the offline phase, each client uniformly distributes t seeds s_{ij} to the first t servers $P_{n-t}, \ldots, P_{n-1}$. The servers expand the seed to OKVS objects and aggregate them to $\vec{S}_j$ as $\vec{S}_j = \bigoplus_{i=1}^{n-t-1} G(s_{ij})$. In the online phase, each client only computes and sends $\vec{S}_{i,n}$ to P_n. Therefore, the online complexity is reduced to $O((n-t)m)$ for P_n and $O(m)$ for clients.

Securely Masking with Pre-generated Triple. The servers complete the mask phase with an mRSM protocol. The mRSM protocol receives a secret-shared vector $[\![\vec{X}]\!]$ as input, and outputs a secret-shared vector $[\![\vec{Y}]\!] = [\![\Delta \cdot \vec{X}]\!]$ where the value Δ is random and can be generated before inputting $[\![\vec{X}]\!]$. We instantiate the mRSM protocol with preprocessing techniques. In the offline phase, parties are required to generate a scalar multiplication triple $([\![\vec{A}]\!], [\![\Delta]\!], [\![\vec{B}]\!])$, where $\vec{B} = \Delta \cdot \vec{A}$. Then, in the online phase, parties can compute $[\![\vec{Y}]\!]$ with a 1-round interaction, solely using symmetric operations as follows:

- **Opening masked vector interactively:** All parties send $\vec{A}'_i = \vec{X}_i \oplus \vec{A}_i$ to a pivot and the pivot broadcasts $\vec{A}'$.
- **Evaluating shares locally:** All parties compute the share $\vec{Y}_i = \Delta_i \cdot \vec{A}' \oplus \vec{B}_i$, satisfying that

$$\bigoplus_i \vec{Y}_i = (\bigoplus_i \Delta_i) \cdot \vec{A}' \oplus \bigoplus_i \vec{B}_i = \Delta \cdot \vec{X}. \quad (7)$$

We note that our online phase achieves linear complexity, that is $O(tm)$ for the pivot, $O(m)$ for others, and $O(tm)$ for total.

To generate such a triple in the offline phase, we propose a triple generation protocol. Our idea is inspired by the observation that $\vec{B}$ can be expanded as

$$\vec{B} = \bigoplus_i \Delta_i \cdot \bigoplus_j \vec{A}_j = \bigoplus_i \left(\Delta_i \cdot \bigoplus_j \vec{A}_j \right). \quad (8)$$

To generate shares of $\vec{B}$, parties are required to generate shares of $\Delta_i \cdot \bigoplus_j \vec{A}_j$. We leverage vector oblivious linear evaluation (VOLE) to achieve the above requirement. VOLE is a secure two-party protocol allowing parties to learn a VOLE correlation, where a receiver learns $(\vec{A}', \vec{B}')$ and a sender learns $(\Delta, \vec{C}' = \vec{A}' \cdot \Delta \oplus \vec{B}')$. Specifically, P_i and P_j first generate a VOLE correlation with the same Δ_i from P_i, where P_i obtains $\vec{C}'_{ij}$ and P_j obtains $(\vec{A}'_{ij}, \vec{B}'_{ij})$ such that $\vec{B}'_{ij} \oplus \vec{C}'_{ij} = \Delta_i \cdot \vec{A}'_{ij}$. Then, P_j masks vector $\vec{A}_j$ with $\vec{A}'_{ij}$, sends the masked vector to P_i, and outputs $\vec{B}'_{ij}$ as the result share. P_i computes the remaining share with the received masked vectors from all other parties. We refer to the detailed protocol in Section 3.

1-round Reconstruction Phase. Once the reducible zero-sharing is completed, the intersection can be efficiently obtained by a 1-round reconstruction protocol. All servers send the masked OKVS objects $\vec{D}'_j$ to the pivot P_n. The pivot aggregates all received OKVS objects as $\vec{D}' = \bigoplus_{j=n-t}^{n} \vec{D}'_j$ and computes $\mathsf{Decode}(\vec{D}', x^n_k)$ for every item $x^n_k \in X_n$. For an item x in the intersection, $\mathsf{Decode}(\vec{D}', x)$ equals zero; otherwise a random value. In this way, the pivot can distinguish and obtain the elements in the intersection with linear complexity. The topology of our construction is presented in Figure 1.

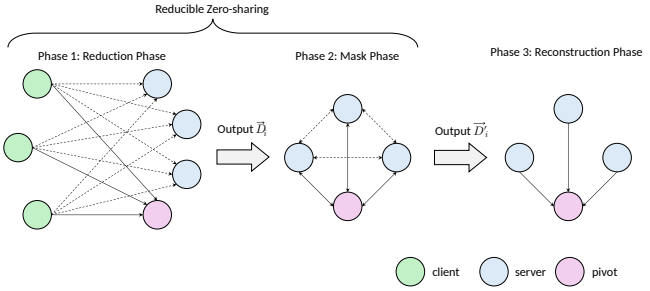


Figure 1. The topology of our construction

Complexity Analysis. Our mPSI protocol offloads costly public-key operations to an offline phase, and achieves a practical online phase, solely using symmetric operations.

Specifically, in the offline phase, parties generate two kinds of correlated randomness:

- **For the reduction phase:** Each client uniformly distributes a seed of length κ to each server except the pivot, resulting in $O(t\kappa)$ communication complexity. After receiving seeds from $n-t-1$ clients, each server

expands the seeds to OKVS objects of size $O(m\lambda)$ and aggregates them, incurring $O((n-t)\kappa)$ communication cost and $O((n-t)m\lambda)$ symmetric operations.
- **For the mask phase:** Each server performs VOLE with all other servers to obtain VOLE correlations, which introduce $t \cdot \mathsf{VOLE}$ complexity. They then mask a vector using the VOLE correlations, send the masked vectors to other servers, and aggregate all received masked vectors to a scalar multiplication triple, with additional $O(tm\lambda)$ computational and communication complexity.

In the online phase, our protocol executes the mPSI computation in three phases, while the reduction phase and the mask phase constitute our reducible zero-sharing construction:

- **The reduction phase:** Each client encodes an OKVS object with $O(m\lambda)$ computational complexity, and sends it to the pivot, resulting in $O(m\lambda)$ communication cost. Afterwards, each server decodes and re-encodes an OKVS object. For the pivot, it incurs $O((n-t)m\lambda)$ computational and communication complexity, while $O(m\lambda)$ computational complexity for other servers.
- **The mask phase:** Each server masks the OKVS object with the triple and sends it to the pivot, with $O(m\lambda)$ complexity. The pivot opens the masked OKVS object to servers, resulting in $O(tm\lambda)$ complexity. They then compute the shares locally, which introduces $O(m\lambda)$ computational complexity.
- **The reconstruction phase:** Each server sends the masked OKVS object to the Leader, incurring $O(m\lambda)$ communication cost for the server and $O(tm\lambda)$ communication cost for the Leader. The pivot then aggregates all received masked OKVS vectors and computes the intersection, resulting in $O(tm\lambda)$ computational cost.

Overall, our protocol achieves:

- **Offline phase:** Linear computational and communication complexity $O(t\kappa)$ for clients and $O(tm\lambda + t\lambda \log m))$ for servers. The overall complexity (sum of all parties) is $O((n-t)t\kappa + t^2(m\lambda + \lambda \log m))$.
- **Online phase:** Computational and communication complexity $O(m\lambda)$ for clients and servers except the pivot, only dependent on m, and linear complexity $O((n+t)m\lambda)$ for the pivot. The overall complexity is linear, that is $O((n+t)m\lambda)$.

This makes our construction significantly practical for mPSI tasks. The asymptotic complexity comparison with existing works is given in Table 1. Our construction achieves a linear total online communication and computational complexity. Compared to works that resist arbitrary collisions [7], [14], both our online communication and computational complexity achieve the best performance, while our overall complexity (offline+online) also outperforms these works. In addition, compared to works with practical efficiency [10], [11], [15], while they fail to achieve sufficient security, our construction achieves similar performance.

1.2. Related Works

Since mPSI was first proposed by Freedman et al. [19], numerous works [6], [9], [12], [13], [20], [21] have focused on the performance improvement of mPSI. Below, we review the works most relevant to ours while the complexity of various mPSI protocols is already listed in Table 1

Directly implementing mPSI. Kolesnikov et al. [7] proposed efficient mPSI protocols of two versions, one against semi-honest adversaries and another against augmented semi-honest adversaries. These protocols primarily follow their newly proposed mPSI paradigm, called the zero-sharing+reconstruction paradigm, built upon their new cryptographic primitive called Oblivious Programmable Pseudorandom Function (OPPRF). They instantiated the OPPRF with three approaches, including garbled bloom filter, polynomial, and hash tables.

In 2024, Wu et al. [14] proposed an mPSI protocol, following the zero-sharing+reconstruction paradigm, where the key idea is the OKVS chain. When one party is honest, they constructed an efficient mPSI protocol with the OKVS chain. To identify the honest party, they utilized Oblivious Pseudorandom Function (OPRF) protocol to improve security. For corruption threshold t, each party is required to perform OPRF protocol with the last t parties, leading to a computation complexity of $O(tm\kappa)$. As for now, their protocols are regarded as the most efficient mPSI protocols capable of resisting arbitrary collisions with available implementation.

Recently, Dong et al. [21] have proposed a framework for multi-party private set operations (mPSO) and its variants, including mPSI. The framework is built upon their proposed primitive, predicative zero-sharing, in which parties obtain shares of zero if a predicative formula evaluates to true, and shares of a random value otherwise. While this framework enables versatile private set operations, the provided instantiations incur relatively high computational and communication costs and do not support optimizations for small corruption thresholds.

Reducing mPSI to PSI of fewer parties. In 2021, Nevo et al. [11] proposed a new mPSI paradigm. In this paradigm, parties reduce their input set to fewer parties, which are required to perform an expensive PSI protocol afterwards. They instantiated the PSI protocol with $\mathsf{ZeroXOR}$ protocol and OPPRF by directly utilizing the unconditional zero-sharing protocol in [7]. While practical in efficiency, their protocol is vulnerable to collusion attacks.

Following this paradigm, the state-of-the-art work is [15] by Gao et al. in 2025. They reduced the mPSI protocol to a two-party PSI protocol, using lightweight symmetric operations. Depending on the significant performance of the two-party PSI protocol, their construction achieves considerable efficiency and low communication cost. However, they heavily relied on the non-collusion assumption that two specific parties are not allowed to collude.

Table 1. COMPARISON OF RELATED WORK AND OUR WORK ON COMMUNICATION AND COMPUTATION COMPLEXITY. n IS THE NUMBER OF PARTIES. t IS THE CORRUPTION THRESHOLD. m IS THE SIZE OF EACH PARTY'S SET. λ AND κ ARE STATISTICAL AND COMPUTATIONAL SECURITY PARAMETERS, RESPECTIVELY. THE SIGN * IMPLIES THAT SPECIFIC PARTIES ARE NOT ALLOWED TO COLLUDE.

Scheme		Communication			Computation			Corruption Threshold	Security Model	Arbitrary Collusion
		Leader	Client	Total (all parties)	Leader	Client	Total (all parties)			
CDG+21 [10]		$O(nm(\lambda+\kappa+\log m))$	$O(m(\lambda+\kappa+\log m))$	$O(nm(\lambda+\kappa+\log m))$	$O(nm\kappa)$	$O(m\kappa)$	$O(nm\kappa)$	$t<\lfloor(n+1)/2\rfloor$	semi-honest	✗
NTY21 [11]		$O(nm\lambda+tm(\kappa+\log m))$	$O(t\kappa+m\lambda)$	$O((n-t)t\kappa+nm\lambda+tm(\lambda+\kappa+\log m))$	$O(nm\lambda)$	$O(m\lambda)$	$O(nm\lambda)$	$t<n$	semi-honest	✗
		$O((n+t)m\kappa+tm\log m)$	$O((m+t)\kappa$	$O((n-t)t\kappa+(n+t)m\kappa+tm\log m)$	$O(nm\kappa)$	$O(m\kappa)$	$O(nm\kappa)$	$t<n$	malicious	✗
GLW+24 [15]		$O(n\kappa+m\lambda+\lambda\log m))$	$O(\kappa+m\lambda)$	$O(n\kappa+nm\lambda+\lambda\log m))$	$O(nm\lambda)$	$O(m\lambda)$	$O(nm\lambda)$	$t<n^*$	semi-honest	✗
		$O((n+m)\kappa+\lambda\log m)$	$O(m\kappa)$	$O((n+nm)\kappa+\lambda\log m)$	$O(nm\kappa)$	$O(m\kappa)$	$O(nm\kappa)$	$t<n-1^*$	malicious	✗
KMP+17 [7]		$O(nm(\lambda+\kappa+\log m))$	$O(m(\lambda+\kappa+\log m))$	$O(nm(\lambda+\kappa+\log m))$	$O(nm\kappa)$	$O(m\kappa)$	$O(nm\kappa)$	$t<n$	augmented semi-honest	✗
			$O(tm(\lambda+\kappa+\log m))$	$O(tnm(\lambda+\kappa+\log m))$		$O(tm\kappa)$	$O(tnm\kappa)$		semi-honest	✓
WYC24 [14]		$O(nm(\lambda+\kappa+\log m))$	$O(tm(\lambda+\kappa+\log m))$	$O(tnm(\lambda+\kappa+\log m))$	$O(nm\kappa)$	$O(tm\kappa)$	$O(tnm\kappa)$	$t<n$	semi-honest	✓
Ours	Offline	$O(tm\lambda+t\lambda\log m))$	$O(t\kappa)$	$O((n-t)t\kappa+t^2(m\lambda+\lambda\log m))$	$O(tm\lambda)$	$O(tm\lambda)$	$O(tnm\lambda)$			
	Online	$O((n+t)m\lambda)$	$O(m\lambda)$	$O((n+t)m\lambda)$	$O((n+t)m\lambda)$	$O(m\lambda)$	$O((n+t)m\lambda)$	$t<n$	semi-honest	✓
	Offline+Online	$O((n+t)m\lambda+t\lambda\log m)$	$O(t\kappa+m\lambda)$	$O((n-t)t\kappa+(n+t+t^2)m\lambda+t^2\lambda\log m)$	$O((n+t)m\lambda)$	$O(tm\lambda)$	$O(tnm\lambda)$			

1.3. Our Contributions

We provide an answer to the aforementioned question by constructing an mPSI protocol with an offline phase. Our construction fully leverages the homomorphic property of the oblivious key-value store (OKVS). We summarize our main contributions as follows.

Reducible Zero-sharing (Section 3). We abstract a cryptographic primitive called reducible zero-sharing $\mathcal{F}_{\mathsf{rzs}}^{n,t,m}$, which allows n parties to securely share their input set to $t+1$ parties, while resisting up to t colluding parties. We propose an efficient construction in the offline/online paradigm, leveraging the homomorphic properties of OKVS. In our construction, we use a novel approach to resist collusion attacks by secure scalar multiplication. We design a secure mRSM protocol using VOLE. The online phase of our construction solely uses symmetric operations and achieves linear total computational and communication complexity. Finally, we prove that our construction $\Pi_{\mathsf{rzs}}^{n,t,m}$ is secure against up to t semi-honest colluding parties.

An Efficient mPSI Protocol (Section 4). Based on a reducible zero-sharing scheme $\Pi_{\mathsf{rzs}}^{n,t,m}$, we propose an efficient mPSI protocol that resists arbitrary collusions when the corruption threshold is $t < n$. Similar to our construction of reducible zero-sharing, our mPSI protocol also works in the offline/online paradigm. By offloading costly operations to the offline phase, the online phase achieves linear total computational and communication complexity, i.e., $O((n+t)m\lambda)$. This enables our mPSI protocol to outperform the existing works and to support a large number of parties as well as high corruption threshold.

Evaluation (Section 5). We implement and make a sufficient benchmark on our mPSI construction. Our mPSI construction is compared with two state-of-the-art (SOTA) works, including [14] and [15]. Compared with the work with sufficient security [14], the experiments show that the online phase of our mPSI construction achieves $9.57-114.46\times$ speedup in the LAN setting, $2.69-28.41\times$ speedup in the WAN setting, and requires up to $28.70\times$ less communication cost. Furthermore, our overall construction (offline+online) outperforms the SOTA work, that is $0.37-18.73\times$ speedup in the LAN setting, $0.76-4.01\times$ speedup in the WAN setting, and requires up to $5.57\times$ less communication cost. Compared with the work with efficiency [15], which relies on the non-collusion assumption, while we achieve security against arbitrary collusion, our construction achieves similar performance, such that $1.52-26.03\times$ speedup in the LAN setting, $0.49-4.87\times$ speedup in the WAN setting, and $0.37-2.26\times$ less communication cost. Moreover, since the implementation of [21] is not yet publicly available, we provide a comparison only for selected cases. Our construction achieves $3.67-8.67\times$ speedup in the LAN setting, $2.34-5.20\times$ speedup in the WAN setting, and $1.25-1.65\times$ less communication cost. More details can be found in Appendix C. Our implementation is available on GitHub.

2. Preliminaries

2.1. Notation

Let λ be the statistical security parameter and κ be the computational security parameter. There are n parties engaging in mPSI. The i-th party, denoted as P_i, inputs a set X_i of size m. The j-th item in set X_i is denoted as x_j^i. We use t to denote the corruption threshold, which is the maximum number of corrupted parties. We denote row vectors $\vec{A}=(a_1,\ldots,a_m)$ using the arrow notation while the elements are indexed without it. $\langle\vec{A},\vec{B}\rangle$ denotes the inner product of $\vec{A},\vec{B}$. The notation $a\oplus b$ means bitwise-XOR of a,b, and the notation $a\cdot b$ refers to multiplication of a,b in the corresponding ring. We use $k\cdot\vec{A}$ to denote the scalar multiplication of k and $\vec{A}$ such that $k\cdot\vec{A}=(ka_1,\ldots,ka_m)$. We use $X\overset{c}{\equiv}Y$ to denote that X is computationally indistinguishable from Y. $a\leftarrow_R A$ denotes that a is uniformly sampled from distribution A. We use $[\![a]\!]$ (resp, $[\![\vec{A}]\!]$) to denote that the element (resp, vector) is shared by parties in boolean secret sharing. The notation $a_i,\vec{A}_i$ denotes the share held by P_i.

2.2. Security Model

In this work, we consider the semi-honest model, where adversaries strictly follow the protocol but attempt to learn additional knowledge through the protocol transcripts. We note that such a model is vulnerable to malicious adversaries, which may lead to limited application. We believe that some realistic scenarios do correspond to semi-honest behaviour. One such example is secure computation among several companies, which are trusted to act honestly due to contact, but cannot fully guarantee that their storage might not be compromised in the future.

We use the real/ideal paradigm from [22] to prove the semi-honest security, with the specific definition given below.

Definition 1. Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-ary functionality, where $f_i(x_1, \ldots, x_n)$ denotes the i-th element of $f(x_1, \ldots, x_n)$. For $I = \{i_1, \ldots, i_t\} \subseteq \{1, \ldots, n\}$, we let $f_I(x_1, \ldots, x_n)$ denote the subsequence $f_{i_1}(x_1, \ldots, x_n), \ldots, f_{i_t}(x_1, \ldots, x_n)$. Let Π be an n-party protocol for computing f. The view of the i-th party during an execution of Π on $\overline{x} = (x_1, \ldots, x_n)$ is denoted by $\mathsf{View}_i^{\Pi}(\overline{x})$. We let $\mathsf{View}_I^{\Pi}(\overline{x}) = (I, \mathsf{View}_{i_1}^{\Pi}(\overline{x}), \ldots, \mathsf{View}_{i_t}^{\Pi}(\overline{x}))$. We say that Π privately computes f if there exists an probabilistic polynomial-time algorithm, denoted Sim, such that for every $I \subseteq \{1, \ldots, n\}$, it holds that

$$\begin{aligned} &\{\mathsf{Sim}(I, (x_{i_1}, \ldots, x_{i_t}), f_I(\overline{x}), f(\overline{x}))\}_{\overline{x} \in (\{0,1\}^*)^n} \\ &\stackrel{c}{\equiv} \left\{(\mathsf{View}_I^{\Pi}(\overline{x}), \mathsf{Output}^{\Pi}(\overline{x}))\right\}_{\overline{x} \in (\{0,1\}^*)^n} \end{aligned} \tag{9}$$

where $\mathsf{Output}^{\Pi}(\overline{x})$ denotes the output sequence of all parties during the execution represented in $\mathsf{VIEW}_I^{\Pi}(\overline{x})$.

2.3. Multi-party Private Set Intersection

The ideal functionality of mPSI $\mathcal{F}_{\mathsf{mpsi}}^{n,m}$ is presented in Figure 2. In mPSI with n parties, each party $P_i (i = 1, \ldots, n)$ inputs a private set $X_i = \left\{x_j^i \in \{0,1\}^* \mid j \in [1, m]\right\}$. They want to compute the intersection without any leakage. In our definition, only P_n finally obtains the intersection.

PARAMETERS: The number of parties n, and the size of each party's set m.
FUNCTIONALITY:
- Wait for input $X_i = \left\{x_1^i, \ldots, x_m^i\right\}$ from party P_i.
- Give output $\bigcap_{i=1}^n X_i$ to party P_n.

Figure 2. The Ideal Functionality of mPSI $\mathcal{F}_{\mathsf{mpsi}}^{n,m}$

2.4. Oblivious Key-value Store

An Oblivious Key-Value Store (OKVS) is a data structure to encode a set of key-value points $A = \{(k_i, v_i)\}$ while hiding the keys. A party can query k on the data structure and obtain v_i if $k = k_i$ or otherwise, a random value.

Definition 2 (OKVS [5], [18])**.** An OKVS scheme is comprised of two algorithms Encode and Decode, parameterized by a set $\mathcal{K}$ of keys, a set $\mathcal{V}$ of values, and a set of functions H, with the following syntax:

- $(S, \perp) \leftarrow \mathsf{Encode}_H(\{(k_i, v_i)\})$: On input a set of key-value points $\{(k_i, v_i)\}$, the encode algorithm outputs an object $S \in \mathcal{S}$, or $\perp$ with a negligible probability.
- $y \leftarrow \mathsf{Decode}_H(S, k)$: On input S and a key $k \in \mathcal{K}$, the decode algorithm outputs a value $v \in \mathcal{V}$.

In the rest of the paper, the underlying parameter H is omitted as long as the text remains unambiguous. For abbreviation, we use $\mathsf{Decode}(S, K)$ to represent the set $\{\mathsf{Decode}(S, k_i) | k_i \in K\}$ and $\mathsf{Encode}(K, V)$ to represent $\mathsf{Encode}(\{(k_i, v_i)\})$ where K and V have the same size. The scheme satisfies

- **Correctness:** For all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys, if $(k, v) \in A$ and $\perp \neq S \leftarrow \mathsf{Encode}(A)$, then $\mathsf{Decode}(S, k) = v$.
- **Doubly obliviousness:** For all distinct $\{k_1, \ldots, k_n\} \in \mathcal{K}^n$ and n values $v_1, \ldots, v_n$ each sampled uniformly at random from $\mathcal{V}$ such that Encode does not output $\perp$ for $\{(k_1, v_i)\}$, $\mathsf{Encode}(\{(k_i, v_i)\})$ is statistically indistinguishable from an uniformly random element in $\mathcal{S}$.

A binary OKVS is a special case of OKVS, where $\mathcal{V} = \mathbb{F} = GF(2^l)$, the output of Encode is a vector $\vec{S}$ in $\mathbb{F}^m$, and the Decode function is defined as $\mathsf{Decode}(\vec{S}, k) = \langle \mathsf{d}(k), \vec{S} \rangle$ for some function $\mathsf{d} : \mathcal{K} \to \{0,1\}^l$. The binary OKVS holds two homomorphic properties. Let $\vec{S} = (s_1, \ldots, s_m) \in \mathbb{F}^m$, then we have the properties that:

- **Additive homomorphism:** Given OKVS objects $\vec{S}, \vec{S'}$ with the same dimension, then define $\vec{S} \oplus \vec{S'} = (s_1 \oplus s'_1, \ldots, s_w \oplus s'_m)$. Then we have

$$\mathsf{Decode}(\vec{S}, x) \oplus \mathsf{Decode}(\vec{S'}, x) = \mathsf{Decode}(\vec{S} \oplus \vec{S'}, x) \tag{10}$$

- **Scalar multiplicative homomorphism:** Given an OKVS object $\vec{S}$ and $b \in \mathbb{F}$, define $b \cdot \vec{S} = (b \cdot s_1, \ldots, b \cdot s_m)$. Then we have

$$\mathsf{Decode}(b \cdot \vec{S}, x) = b \cdot \mathsf{Decode}(\vec{S}, x) \tag{11}$$

2.5. Vector Oblivious Linear Evaluation

Vector Oblivious Linear Evaluation (VOLE) [23] enables a sender to learn a secret linear combination of two vectors held by a receiver. In this work, we focus on the random variant of VOLE, which we continue to refer to as VOLE for simplicity. Let $\mathbb{F}$ be some field. In VOLE, the parties input nothing. The Sender obtains a random value $\Delta \in \mathbb{F}$ and a random vector $\vec{C} \in \mathbb{F}^m$, while the Receiver obtains two random vectors $\vec{A}, \vec{B} \in \mathbb{F}^m$. We further consider a special case of VOLE where the Sender can input the random value Δ. It has been efficiently implemented by a cryptographic primitive called pseudorandom correlation generator (PCG)

in recent work [24]. It allows parties to generate instances of VOLE with the same Δ. We present the ideal functionality of VOLE $\mathcal{F}_{\mathsf{vole}}^{\mathbb{F},m}$ in Figure 3.

PARAMETERS: Two parties, a Sender and a Receiver, the field $\mathbb{F}$, and the size of the output vector m.
FUNCTIONALITY:
- Wait for input $(\mathsf{sender}, \mathsf{sid}, \Delta \in \mathbb{F})$ from the Sender and $(\mathsf{receiver}, \mathsf{sid})$ from the Receiver.
- Sample $\vec{A}, \vec{B} \in \mathbb{F}^m$ and compute $\vec{C} = \vec{A} \cdot \Delta \oplus \vec{B}$.
- Give $(\vec{C}, \Delta)$ to the Sender and $(\vec{A}, \vec{B})$ to the Receiver.

Figure 3. The Ideal Functionality of VOLE $\mathcal{F}_{\mathsf{vole}}^{\mathbb{F},m}$

3. Reducible Zero-sharing

In this work, we abstract a new cryptographic primitive called reducible zero-sharing, which plays a main building block in our mPSI protocol, ensuring our construction is secure against collusion attacks. Specifically, in an (n,t)-reducible zero-sharing $\mathcal{F}_{\mathsf{rzs}}^{n,t,m}$, each party $P_i (i \in [1,n])$ inputs a set X_i of size m. Then only $t+1$ parties obtain the output, where each of the $t+1$ parties outputs a mapping f_i such that for item x in the intersection, the mappings hold $\bigoplus f_i(x) = 0$. The ideal functionality of (n,t,m)-reducible zero-sharing $\mathcal{F}_{\mathsf{rzs}}^{n,t,m}$ is shown in Figure 4.

PARAMETERS: The number of parties n, the corruption threshold t, and the size of set m.
FUNCTIONALITY:
- Wait for input X_i from party P_i, where $|X_i| = m$.
- Sample $t+1$ mappings $f_i (i \in [1, t+1])$ such that
$$\bigoplus_{i=1}^{t+1} f_i(x) = 0 \text{ for every } x \in \bigcap_{i=1}^{n} X_i$$
- Sample $t+1$ parties to form a set $\mathcal{P}$.
- Give mapping f_i to $\mathcal{P}_i, i \in [1, t+1]$.

Figure 4. The Ideal Functionality of (n,t)-Reducible Zero-sharing $\mathcal{F}_{\mathsf{rzs}}^{n,t,m}$

3.1. Subprotocol: Secure Multi-party Random Scalar Multiplication

Secure multi-party random scalar multiplication (mRSM) protocol is a core subprotocol in our construction of reducible zero-sharing. It allows parties to securely perform a scalar multiplication on a secret-shared vector with a random scalar. Specifically, in an n-party mRSM protocol, each party P_i inputs a secret-shared vector $\vec{X}_i$. Then they obtain a secret-shared vector $\vec{Y}_i$, which is multiplied by a random scalar Δ as $\vec{Y} = \Delta \cdot \vec{X}$, where $\vec{Y} = \bigoplus \vec{Y}_i$ and $\vec{X} = \bigoplus \vec{X}_i$.

We leverage VOLE to design a triple generation protocol $\Pi_{\mathsf{tg}}^{n,m}$, constructing the offline phase of mRSM protocol. Parties first generate shares $[\![\vec{A}]\!]$ and $[\![\Delta]\!]$. Then, each pair of parties (P_i, P_j) first generate a VOLE correlation, where P_i obtains $(\Delta_i, \vec{C}'_{ij})$ and P_j obtains $(\vec{A}'_{ij}, \vec{B}'_{ij})$ such that $\vec{B}'_{ij} \oplus \vec{C}'_{ij} = \Delta \cdot \vec{A}'_{ij}$. Afterwards, P_j masks $\vec{A}_j$ with $\vec{A}'_{ij}$ as $\vec{A}^*_{ij} = \vec{A} \oplus \vec{A}'_{ij}$ and sends $\vec{A}^*_{ij}$ to P_i. Finally, each party aggregates all received vectors to obtain shares of $\Delta \cdot \vec{A}$. We present the detailed construction of $\Pi_{\mathsf{tg}}^{n,m}$ in Figure 5.

PARAMETERS: Field $\mathbb{F}$, the number of parties n, the length of vector m, a VOLE scheme $\mathcal{F}_{\mathsf{vole}}$.
PROTOCOL:
1) P_i randomly samples $\vec{A}_i \leftarrow_R \mathbb{F}^m, \Delta_i \leftarrow_R \mathbb{F}$.
2) P_i and P_j engage in $\mathcal{F}_{\mathsf{vole}}^{\mathbb{F},m}$, where P_i acts as the Sender with input Δ_i and P_j acts as the Receiver.
   - In each execution, P_i obtains $\vec{C}'_{ij}$ and P_j obtains $\vec{A}'_{ij}, \vec{B}'_{ij} = \Delta_i \cdot \vec{A}'_{ij} \oplus \vec{C}'_{ij}$.
3) P_j computes $\vec{A}^*_{ij} = \vec{A}_j \oplus \vec{A}'_{ij}$ and sends $\vec{A}^*_{ij}$ to P_i.
4) P_i computes $\vec{B}_i = (\vec{A}_i \oplus \bigoplus_{j \in [1,n] \setminus \{i\}} \vec{A}^*_{ij}) \cdot \Delta_i \oplus \bigoplus_{j \in [1,n] \setminus \{i\}} \vec{C}'_{ij} \oplus \bigoplus_{j \in [1,n] \setminus \{i\}} \vec{B}'_{ji}$.
5) P_i outputs triple $(\vec{A}_i, \Delta_i, \vec{B}_i)$.

Figure 5. Triple Generation Protocol for mRSM Offline Phase $\Pi_{\mathsf{tg}}^{n,m}$

Once parties obtain the triple, they can complete the secure scalar multiplication in the following steps. First, parties interactively open a masked vector $\vec{X}' = \vec{X} \oplus \vec{A}$. Afterwards, they can compute shares of $\vec{Y} = \Delta \cdot \vec{X}$ with the masked vector locally as evaluating $\vec{Y}_i = \Delta_i \cdot \vec{X}' \oplus \vec{B}_i$. We present the detailed construction of $\Pi_{\mathsf{mrsm}}^{n,m}$ in Figure 6.

PARAMETERS: Field $\mathbb{F}$, the number of parties n, the length of vector m, and the triple $(\vec{A}_i, \Delta_i, \vec{B}_i)$.
INPUTS: P_i inputs $\vec{X}_i \in \mathbb{F}^m$
PROTOCOL:
1) P_i sends $\vec{X}'_i = \vec{X}_i \oplus \vec{A}_i$ to P_n.
2) P_n sends $\vec{X}' = \bigoplus_{i=1}^{n} \vec{X}'_i$ to all others.
3) P_i computes share $\vec{Y}_i = \Delta_i \cdot \vec{X}' \oplus \vec{B}_i$.

Figure 6. Secure Multi-party Random Scalar Multiplication Protocol for mRSM Online Phase $\Pi_{\mathsf{mrsm}}^{n,m}$

3.2. Protocol Details

Our construction of reducible zero-sharing consists of two phases: a reduction phase and a mask phase. In the

reduction phase, clients share their input sets to servers in OKVS objects, while servers mask the obtained OKVS objects in the mask phase.

To optimize these two phases, we propose a construction of (n, t, m)-reducible zero-sharing protocol in the offline/online paradigm. In the offline phase, parties generate some correlated randomness, while they follow an efficient reduction and subsequent mask phase in the online phase, using the correlated randomness.

Offline Phase. In the offline phase, parties engage in two secure correlated randomness generation procedures: OKVS seed generation for the reduction phase, and triple generation for the mask phase.

- **OKVS seed generation.** each client uniformly picks a random seed $seed_{ij}$ and sends $seed_{ij}$ to server $P_j(j \in [n-t, n))$. They then expand the seeds to OKVS objects and aggregate these objects to $\vec{S}_i$.
- **triple generation.** The servers engage in a triple generation protocol $\Pi_{\mathsf{tg}}^{t+1,m}$ and obtain a triple $(\vec{A}_i, \Delta_i, \vec{B}_i)$.

Online Phase. In the online phase, parties engage in an efficient protocol with the correlated randomness generated in the offline phase.

In the reduction phase, each client performs an OKVS decoding and encoding on its input set with the OKVS object generated in the offline phase and sends the result OKVS object $\vec{S}_{i,n}$ to server P_n. After receiving all OKVS objects from clients, P_n aggregates them to $\vec{S}_n$. Finally, each server P_j performs an OKVS decoding and encoding, computing $\vec{D}_j = \mathsf{Encode}(X_j, \mathsf{Decode}(\vec{S}_j, X_j))$.

In the mask phase, servers input $\vec{D}_j$ to the mRSM protocol $\Pi_{\mathsf{mrsm}}^{t+1,m}$, with triple $(\vec{A}_i, \Delta_i, \vec{B}_i)$ to obtain $\vec{D}'_j$.

We present our detailed construction in Figure 7.

A Special Case. We note that when $t = n - 1$, there is no client in our reducible zero-sharing construction, and servers cannot obtain the initial OKVS objects from the reduction phase, which makes the following phase infeasible. In this case, each party P_i is required to collaboratively generate OKVS objects $\vec{S}_i$. As the property of OKVS objects generated in the OKVS seed generation, these objects satisfy that for $x \in \mathcal{K}$, $\bigoplus_{i=1}^{n} \mathsf{Decode}(\vec{S}_i, x) = 0$.

We note that this can be directly done by a generic MPC, such as the BMR protocol [25], while a custom approach is to leverage the unconditional zero-sharing [7].

In this paper, we do not implement these two approaches. In the experiment, we use a TTP to simulate the OKVS seed generation in this case.

3.3. Security Proof

Theorem 1. In the $\mathcal{F}_{\mathsf{vole}}^{\mathbb{F},m}$-hybrid model, $\Pi_{\mathsf{rzs}}^{n,t,m}$ in Figure 7 securely computes $\mathcal{F}_{\mathsf{rzs}}^{n,t,m}$ against $t < n$ colluding semi-honest parties.

Proof. Our proof includes correctness and security.

PARAMETERS: The number of parties n, the corruption threshold t, the set size m, a binary OKVS Scheme $(\mathsf{Encode}, \mathsf{Decode})$, a pseudorandom generator G, a multi-party scalar multiplication triple generation protocol $\Pi_{\mathsf{tg}}^{t+1,m}$, and a multi-party secure scalar multiplication protocol $\Pi_{\mathsf{mrsm}}^{t+1,m}$.
INPUTS: P_i inputs X_i, where $|X_i| = m$

Offline Phase:

1) **[OKVS Seed Generation]**
   a) $P_i(i \in [1, n-t))$ generates t seeds $s_{ij} \leftarrow_R \{0,1\}^\kappa$ and sends s_{ij} to P_j for $j \in [n-t, n)$.
   b) $P_i(i \in [1, n-t))$ computes $\vec{S}_i$ as $\vec{S}_i = \bigoplus_{j=n-t}^{n-1} G(s_{ij})$.
   c) $P_j(j \in [n-t, n))$ computes $\vec{S}_j$ as $\vec{S}_j = \bigoplus_{i=1}^{n-t-1} G(s_{ij})$.
2) **[Triple Generation]**
   a) $P_j(j \in [n-t, n])$ performs $\Pi_{\mathsf{tg}}^{t+1,m}$ and receives triple $(\vec{A}_i, \Delta_i, \vec{B}_i)$.

Online Phase:

1) **[Reduction Phase]**
   a) $P_i(i \in [1, n-t))$ sends $\vec{S}_{i,n}$ to P_n, where $\vec{S}_{i,n} = \mathsf{Encode}(X_i, \mathsf{Decode}(\vec{S}_i, X_i))$
   b) P_n computes $\vec{S}_n = \bigoplus_{i=1}^{n-t-1} \vec{S}_{i,n}$.
   c) $P_j(j \in [n-t, n])$ computes $\vec{D}_j = \mathsf{Encode}(X_j, \mathsf{Decode}(\vec{S}_j, X_j))$.
2) **[Mask Phase]**
   a) $P_j(j \in [n-t, n])$ engages in $\Pi_{\mathsf{mrsm}}^{t+1,m}$ with input $\vec{D}_j$ and triple $(\vec{A}'_i, \Delta_j, \vec{B}'_j)$, and obtains $\vec{D}'_j$.

Figure 7. Our (n, t, m)-Reducible Zero-sharing Protocol $\Pi_{\mathsf{rzs}}^{n,t,m}$

Correctness. We first prove that the triple generated in $\Pi_{\mathsf{tg}}^{n,m}$ is correct. According to the functionality of VOLE, we have

$$\vec{A}^*_{ij}\Delta_i \oplus \vec{C}'_{ij} = (\vec{A}_j \oplus \vec{A}'_{ij})\Delta_i \oplus \vec{C}'_{ij} = \vec{A}_j\Delta_i \oplus \vec{B}'_{ij}. \quad (12)$$

Therefore, We simplify $\vec{B}_i$ as

$$\vec{B}_i = \bigoplus_{j=1}^{n} \vec{A}_j\Delta_i \oplus \bigoplus_{j\in[1,n]\setminus\{i\}} (\vec{B}'_{ij} \oplus \vec{B}'_{ji}) \quad (13)$$

Then, the XOR-sum of $\vec{B}_i$ equals $\vec{A} \cdot \Delta$. Therefore, the correctness of $\Pi_{\mathsf{tg}}^{n,m}$ holds.

We now prove that with the triple, $\Pi_{\mathsf{mrsm}}^{n,m}$ correctly implements the secure mRSM. We prove that the XOR-sum of $\vec{Y}_i$ is actually $\Delta \cdot \vec{X}$. The XOR-sum of $\vec{Y}_i$ holds that

$$\bigoplus_{i=1}^{n} \vec{Y}_i = \bigoplus_{i=1}^{n} \Delta_i \cdot \vec{X}' \oplus \bigoplus_{i=1}^{n} \vec{B}_i = \Delta \cdot \vec{X}' \oplus \Delta \cdot \vec{A} = \Delta \cdot \vec{X}. \quad (14)$$

Therefore, the correctness of $\Pi_{\mathsf{mrsm}}^{n,m}$ holds.

Finally, we prove that $\Pi_{\mathsf{rzs}}^{n,t,m}$ correctly implements the functionality of $\mathcal{F}_{\mathsf{rzs}}^{n,t,m}$. For an item x in the intersection, according to the homomorphic properties of binary OKVS, we have $\mathsf{Decode}(\vec{D}', x) = \Delta \cdot \bigoplus_{j=n-t}^{n} \mathsf{Decode}(\vec{D}_j, x)$. Since x is in the intersection, it holds that $\mathsf{Decode}(\vec{D}_j, x) = \mathsf{Decode}(\vec{S}_j, x)$. Note that since $\vec{S}_i$ is generated by clients with items as keys and shares of zero as values, it holds that $\mathsf{Decode}(\vec{S}_n, x) = \bigoplus_{i=n-t}^{n-1} \mathsf{Decode}(\vec{S}_i, x)$. Therefore, we have $\mathsf{Decode}(\vec{D}', x) = 0$ for items in the intersection. For an item x excluded from the intersection, there exists an OKVS object $\vec{D}_j$ such that $\mathsf{Decode}(\vec{D}_j, x)$ is a random share, making $\mathsf{Decode}(\vec{D}', x)$ random.

Security. Intuitively, the security of $\Pi_{\mathsf{rzs}}^{n,t,m}$ holds due to the security of VOLE. We refer to the detailed proof in Appendix A.

□

3.4. Complexity Analysis

VOLE serves the main build block in our multi-party secure scalar multiplication construction. As analyzed in [23], a VOLE of size m requires $O(\lambda \log m)$ communication and $O(\lambda m)$ symmetric computation.

Offline Complexity.

- **Clients:** Each client sends a seed of length κ to each server except the pivot. The clients then expand the seeds to OKVS objects and aggregate them to an initial OKVS object. Therefore, the communication cost is $O(t\kappa)$ and the computational cost is $O(tm\lambda)$.
- **Servers:** Each server except the pivot receives $n-t-1$ seeds from clients. They then expand them to OKVS objects and aggregate them. Afterwards, each server engages in a triple generation protocol with $O(t)$ VOLE execution. To generate a triple, they send a masked vector to each other. Overall, for servers except the pivot, the communication complexity is $O((n-t)\kappa + tm\lambda + t\lambda \log m)$ and the computational complexity is $O(tm\lambda)$. For pivot, communication complexity is $O(tm\lambda + t\lambda \log m)$, while the computational complexity is $O(tm\lambda)$.

Online Complexity.

- **Clients:** Each client performs an OKVS decoding and encoding and sends an OKVS object to the pivot, resulting in $O(m\lambda)$ communication complexity and $O(m\lambda)$ computational complexity.
- **Servers:** Each server acts like the client, while they engage in the mask phase where they send and receive a masked OKVS object with pivot, introducing $O(m\lambda)$ communication cost and $O(m\lambda)$ computational cost.
- **Pivot:** The pivot receives $n-t-1$ OKVS objects from clients and t OKVS objects from servers. Additionally, it performs an OKVS decoding and encoding on the aggregation of these $n-t-1$ OKVS objects. In the mask phase, the pivot also aggregates the t OKVS objects to open a masked vector in $\Pi_{\mathsf{mrsm}}^{t+1,m}$. Thus, these lead to $O((n+t)m\lambda)$ communication cost and $O((n+t)nm\lambda)$ computational cost.

4. Multi-party Private Set Intersection

4.1. Our Construction

Our construction works in the offline/online paradigm, leveraging a $(n, t+1, m)$-reducible zero-sharing scheme $\Pi_{\mathsf{rzs}}^{n,t+1,m}$ to resist up to t colluding parties.

Offline Phase. Parties directly invoke the offline phase of $\Pi_{\mathsf{rzs}}^{n,t+1,m}$. Each party, except the pivot, obtains an OKVS object $\vec{S}_i$, while all servers hold a triple $(\vec{A}_i, \Delta_i, \vec{B}_i)$.

Online Phase. In the online phase, parties engage in the online phase of reducible zero-sharing. As a result, the servers obtain masked OKVS objects $\vec{D}'_j$.

Then in a reconstruction phase, all servers send the masked OKVS objects $\vec{D}'_j$ to the pivot. The pivot aggregates them as $\vec{D}' = \bigoplus_{j=n-t}^{n} \vec{D}'_j$ and outputs the intersection.

We present our construction in Figure 8.

PARAMETERS: The number of parties n, a corrupted threshold t, the size of set m, a binary OKVS Scheme $(\mathsf{Encode}, \mathsf{Decode})$, and an (n, t, m)-reducible zero-sharing protocol $\Pi_{\mathsf{rzs}}^{n,t,m}$.
INPUTS: P_i inputs X_i, where $|X_i| = m$
Offline Phase:
1) $P_i(i \in [1, n])$ runs the offline phase of $\Pi_{\mathsf{rzs}}^{n,t,m}$.
2) $P_i(i \in [1, n))$ obtains an initial OKVS object $\vec{S}_i$.
3) $P_j(j \in [n-t, n])$ obtains a triple $(\vec{A}_j, \Delta_j, \vec{B}_j)$.

Online Phase:
1) $P_i(i \in [1, n])$ engages in the online phase of $\Pi_{\mathsf{rzs}}^{n,t,m}$ with X_i, $\vec{S}_i$ and $(\vec{A}_i, \Delta_i, \vec{B}_i)$. As a result, $P_j(j \in [n-t, n])$ obtains $\vec{D}'_j$.
2) $P_j(j \in [n-t, n))$ sends $\vec{D}'_j$ to P_n.
3) P_n computes $\vec{D}' = \bigoplus_{j=n-t}^{n} \vec{D}'_j$ and outputs $\{x_i | \mathsf{Decode}(\vec{D}', x_i) = 0\}$.

Figure 8. Our mPSI Protocol $\Pi_{\mathsf{mpsi}}^{n,m}$

4.2. Security Proof

Theorem 2. In the $\mathcal{F}_{\mathsf{rzs}}^{n,t,m}$-hybrid model, $\Pi_{\mathsf{mpsi}}^{n,m}$ in Figure 8 securely computes $\mathcal{F}_{\mathsf{mpsi}}^{n,m}$ in the presence of any semi-honest adversary corrupting $t < n$ parties.

Proof. Our proof consists of correctness and security.

Correctness. As the correctness of $\Pi_{\mathsf{rzs}}^{n,t,m}$ guarantees, for an item x in the intersection, $\mathsf{Decode}(\vec{D}'_j, x)$ XOR-sum to zero, while otherwise, it is a random value. It can directly derive from the correctness of $\Pi_{\mathsf{mpsi}}^{n,m}$.

Security. Intuitively, since $\Pi_{\mathsf{rzs}}^{n,t,m}$ is secure against up to t colluding parties and the binary OKVS scheme holds the obliviousness property, our construction remains secure while the sole additional operations is opening the masked OKVS objects. We refer to the detailed proof in Appendix B.

□

4.3. Complexity Analysis

We instantiate our construction with the binary OKVS from [5]. Given m key-value pairs, the encode algorithm of a binary OKVS outputs a vector of size $O(m\lambda)$ with $O(m\lambda)$ computational complexity. For decoding, the computational complexity is $O(\lambda)$ per key.

Party Complexity. Since parties directly invoke the offline phase of $\Pi_{\mathsf{rzs}}^{n,t,m}$ in the offline phase, their offline complexity is the same. In the online phase, the additional operation is opening the masked OKVS objects to the pivot, which is $O(tm\lambda)$ cost. Therefore, the online complexity of each party is also the same as the $\Pi_{\mathsf{rzs}}^{n,t,m}$.

Total Complexity. We analyze the total complexity of our construction, i.e., the cost of all parties.

In the offline phase, there are $n-t-1$ clients engaging in the generation of OKVS seeds, introducing $O((n-t)t\kappa)$ communication cost and $O(tnm\lambda)$ computational cost. $t+1$ servers perform triple generation where incurs $O(t^2)$ VOLE execution, resulting in a communication complexity of $O(t^2(m\lambda + \lambda \log m))$ and a computational complexity of $O(tnm\lambda)$. Therefore, the total communication and computational complexity of the offline phase is $O((n-t)t\kappa + t^2(m\lambda + \lambda \log m))$ and $O(tnm\lambda)$, respectively.

In the online phase, the pivot XORs $n+t$ OKVS objects in total and performs an OKVS decoding and encoding, while each server and client performs an OKVS decoding and encoding. This introduces $O((n+t)m\lambda)$ communication cost and $O((n+t)m\lambda)$ computational cost.

5. Experiment

In this section, we try to answer the following questions:

- **Q1:** How does our mPSI construction scale with the number of parties and corruption threshold?
- **Q2:** Does our construction outperform existing works?

To answer the above question, we implement both the offline phase and online phase of our construction in Figure 8 using C++, where each party uses $n-1$ threads to interact simultaneously with all other parties. When $t = n-1$, we use a TTP to only generate correlated randomness for the reduction phase. We instantiate our construction with OKVS from [5] and VOLE from [24]. We use the following libraries in our implementation.

- VOLE: We use implementation in libOTe [26], instantiating the code family with Expand-Convolute codes [24].
- OKVS: We use the binary OKVS implementation in volepsi [27] with field $GF(2^{128})$.

The benchmark machine is with 56-core Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz and 128G RAM. We evaluate our construction in simulated networks using the Linux command tc as other mPSI works. Our network settings are similar to [14], [15]: for the LAN setting, the bandwidth is 10 Gbps and the round-trip latency is 0.02ms; for the WAN setting, the bandwidth is 200 Mbps and the round-trip latency is 96 ms.

Our evaluations are performed under statistical parameter $\lambda = 40$ and computational parameter $\kappa = 128$. We store elements of input sets in a 128-bit string. All results are the average of ten replicates. The communication cost refers to the sum of data that a party sends and receives, while the total communication cost is data that all parties send. The complete implementation of our construction is available on GitHub: https://github.com/Ame-ReIori/mPSI.

5.1. Evaluation of Our Construction

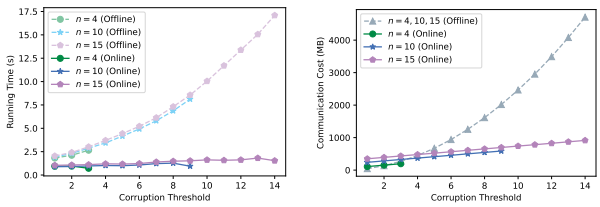
To answer question **Q1**, we evaluate our implementation in two cases: (Case 1) fixing the number of parties while varying the corruption threshold, and (Case 2) fixing the corruption threshold while varying the number of parties. In both cases, we consider $m = 2^{20}$ and a LAN network.

In Case 1, we make a benchmark on the range of $n \in \{4, 10, 15\}$ and all $t < n$. In Figure 9a, we report the total running time of the offline phase and the online phase. We observe that the total running time of the offline phase grows quadratically. This is because each server is required to perform VOLE with other servers, resulting in an overall $O(t^2)$ VOLE executions. In the online phase, the running time grows linearly. We note that the running time when $t = n-1$ is less than that in some cases where $t < n-1$. It is reasonable due to the reduction phase. When $t < n-1$, servers must wait for pivot to finish the reduction phase. In the case where $t = n-1$, parties directly engage in the mask phase, making the protocol faster. In Figure 9b, we report the communication cost of the offline phase and the online phase. Recall, the offline phase of our construction consists of two generations of correlated randomness: (i) one for the reduction phase and (ii) one for the mask phase. In generation (i), clients send some seeds of length κ to servers, resulting in a total communication cost of $(n-t-1)t\kappa$ bits. Its impact on the overall communication cost is negligible, as servers perform $t(t+1)$ executions of VOLE in generation (ii), which leads to a large amount of communication cost that only depends on the corruption threshold t. Therefore, for clarity, we plot the offline performance for $n = 4, 10, 15$ on a single curve. From Figure 9b, we observe that the communication cost of the offline phase grows quadratically, while that of the online phase grows linearly.

In Case 2, we fix the corruption threshold at $t = 3$ and consider $n \in \{5, 15, 25, \ldots, 105\}$. The results are presented in Figure 10. We observe that both the total running time and communication cost scale linearly. In the offline phase, the costly VOLE execution depends only on the corruption threshold t, while the OKVS seed generation is linearly related to the number of parties. Furthermore, the OKVS seed generation introduces minimal communication cost, as clients only send a few seeds to others, resulting in a relatively flat curve in Figure 10b. In the online phase, given a fixed corruption threshold, only the reduction phase depends on the number of parties, which keeps the overall protocol complexity linear.

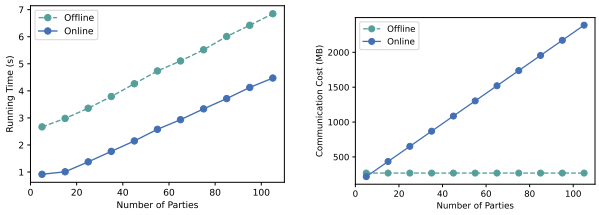
Table 2. THE RUNNING TIME COMPARISONS (IN SECONDS) BETWEEN OUR PROTOCOL AND OTHER MPSI PROTOCOLS. THE BEST RESULT IS MARKED IN BLUE AND THE SECOND BEST RESULT IS MARKED IN ORANGE. CELLS WITH – IMPLY THAT NO PARTIES ACT IN THIS ROLE IN THIS CASE. THE SIGN † IMPLIES THAT WE USE A TTP TO GENERATE THE CORRELATED RANDOMNESS FOR THE REDUCTION PHASE WHEN $t = n - 1$.

Network		LAN										WAN									
Protocol		Oring [14]		Kstar [14]		GLW [15]		Ours				Oring [14]		Kstar [14]		GLW [15]		Ours			
m	(n,t)	Client	Online	Client	Online	Client	Online	Client	Server	Online	Total	Client	Online	Client	Online	Client	Online	Client	Server	Online	Total
2^{12}	$(4,1)$	0.026	0.066	0.026	0.071	0.004	0.080	0.002	0.004	0.005	0.176	0.474	2.126	0.473	2.131	0.246	1.301	0.002	0.347	0.451	2.806
	$(4,2)$	0.069	0.123	0.065	0.115	0.004	0.080	0.002	0.004	0.005	0.182	2.069	3.649	2.068	3.647	0.246	1.301	0.002	0.378	0.457	2.819
	$(4,3)$	0.098	0.129	0.099	0.124	0.004	0.080	–	0.003	0.003	0.216†	3.422	3.647	3.418	3.643	0.246	1.301	–	0.136	0.267	2.693†
	$(10,1)$	0.044	0.147	0.038	0.105	0.005	0.098	0.002	0.005	0.006	0.185	1.309	3.539	1.308	3.562	0.246	1.307	0.002	0.348	0.452	2.808
	$(10,4)$	0.128	0.245	0.128	0.190	0.005	0.098	0.001	0.004	0.005	0.246	3.754	5.693	3.754	5.603	0.246	1.307	0.002	0.360	0.468	2.943
	$(10,8)$	0.135	0.308	0.137	0.254	0.005	0.098	0.001	0.004	0.005	0.399	3.866	5.897	3.869	5.794	0.246	1.307	0.002	0.392	0.495	3.602
	$(10,9)$	0.216	0.344	0.200	0.240	0.005	0.098	–	0.003	0.004	0.438†	5.400	6.030	5.435	5.925	0.246	1.307	–	0.164	0.311	3.688†
	$(15,1)$	0.121	0.333	0.102	0.189	0.007	0.104	0.002	0.005	0.006	0.185	5.561	8.273	5.561	8.314	0.246	1.312	0.002	0.350	0.453	2.808
	$(15,4)$	0.201	0.396	0.235	0.307	0.007	0.104	0.002	0.004	0.005	0.237	9.470	11.74	9.403	11.71	0.246	1.312	0.002	0.372	0.470	2.929
	$(15,7)$	0.281	0.441	0.229	0.311	0.007	0.104	0.001	0.004	0.005	0.355	9.552	11.81	9.647	11.70	0.246	1.312	0.002	0.389	0.489	3.359
	$(15,13)$	0.292	0.593	0.243	0.398	0.007	0.104	0.001	0.005	0.006	0.628	9.752	12.60	9.776	12.42	0.246	1.312	0.002	0.474	0.617	5.818
	$(15,14)$	0.383	0.579	0.371	0.415	0.007	0.104	–	0.003	0.005	0.694†	11.92	12.95	12.04	12.76	0.246	1.312	–	0.209	0.456	6.285†
2^{16}	$(4,1)$	0.374	0.678	0.401	0.679	0.054	0.257	0.030	0.065	0.071	0.364	0.842	3.442	0.850	3.853	0.321	2.625	0.037	0.888	1.213	3.620
	$(4,2)$	0.554	0.922	0.552	0.914	0.054	0.257	0.023	0.059	0.066	0.360	2.563	5.018	2.562	5.023	0.321	2.625	0.036	0.988	1.407	3.924
	$(4,3)$	0.688	1.073	0.707	0.991	0.054	0.257	–	0.037	0.046	0.387†	4.339	5.238	4.342	5.201	0.321	2.625	–	0.616	1.064	4.047†
	$(10,1)$	0.474	1.582	0.531	1.062	0.070	0.278	0.035	0.072	0.077	0.378	1.681	6.867	1.700	7.857	0.321	2.925	0.038	1.139	1.495	3.909
	$(10,4)$	0.721	2.127	0.755	1.278	0.070	0.278	0.027	0.065	0.074	0.429	4.736	9.810	4.786	8.708	0.321	2.925	0.037	1.529	1.922	5.193
	$(10,8)$	0.849	2.183	0.858	1.406	0.070	0.278	0.023	0.072	0.087	0.712	5.629	11.80	5.615	10.42	0.321	2.925	0.037	2.194	3.410	9.246
	$(10,9)$	0.928	2.267	0.976	1.498	0.070	0.278	–	0.050	0.074	0.792†	8.982	12.62	9.166	10.89	0.321	2.925	–	1.327	3.140	10.21†
	$(15,1)$	0.596	2.373	0.602	1.272	0.083	0.302	0.031	0.072	0.078	0.393	5.941	14.10	5.952	14.69	0.326	3.129	0.038	1.388	1.749	4.172
	$(15,4)$	0.927	3.010	0.959	1.648	0.083	0.302	0.028	0.069	0.079	0.470	11.31	19.84	11.52	15.96	0.326	3.129	0.036	1.813	2.163	5.432
	$(15,7)$	1.116	3.264	1.111	1.776	0.083	0.302	0.023	0.074	0.085	0.664	12.60	21.04	12.75	17.76	0.326	3.129	0.037	1.888	2.981	8.171
	$(15,13)$	1.377	3.467	1.518	2.332	0.083	0.302	0.024	0.118	0.132	1.234	13.37	25.54	13.57	22.51	0.326	3.129	0.037	2.729	4.058	18.33
	$(15,14)$	1.528	3.592	1.608	2.285	0.083	0.302	–	0.086	0.133	1.332†	20.66	26.56	21.04	23.51	0.326	3.129	–	2.173	4.957	20.31†
2^{20}	$(4,1)$	10.28	17.74	10.40	15.14	0.701	2.169	0.340	0.743	0.813	2.638	12.07	22.01	13.38	20.57	4.822	9.653	2.911	7.280	7.561	14.45
	$(4,2)$	11.99	19.56	12.16	17.62	0.701	2.169	0.311	0.768	0.855	2.979	20.18	29.75	20.27	28.73	4.822	9.653	2.284	8.750	9.217	19.80
	$(4,3)$	13.46	20.96	13.40	18.21	0.701	2.169	–	0.544	0.650	3.308†	24.76	34.49	25.32	31.90	4.822	9.653	–	8.931	9.582	25.24†
	$(10,1)$	10.85	38.70	11.12	19.87	0.880	2.415	0.357	0.782	0.852	2.749	13.71	51.34	15.33	33.70	9.626	15.08	8.003	12.27	12.54	19.50
	$(10,4)$	14.44	42.80	14.63	23.42	0.880	2.415	0.353	0.934	1.050	4.461	44.50	82.15	50.18	64.34	9.626	15.08	5.682	17.25	17.95	40.96
	$(10,8)$	16.43	44.69	16.49	26.13	0.880	2.415	0.357	1.197	1.321	8.239	80.07	116.3	73.28	88.01	9.626	15.08	2.475	25.49	25.36	97.78
	$(10,9)$	18.28	46.32	18.81	27.50	0.880	2.415	–	0.941	1.105	9.267†	90.80	126.4	91.53	106.0	9.626	15.08	–	23.83	26.01	115.5†
	$(15,1)$	11.31	56.31	11.74	23.58	1.019	2.567	0.442	0.884	0.954	3.006	18.39	78.51	20.46	47.83	13.59	19.58	12.52	16.83	17.12	24.16
	$(15,4)$	16.66	62.21	16.96	28.95	1.019	2.567	0.365	0.972	1.086	4.736	67.53	128.5	74.43	94.46	13.59	19.58	10.10	21.59	22.39	45.51
	$(15,7)$	19.80	65.07	19.95	32.10	1.019	2.567	0.358	1.177	1.305	7.419	98.48	164.8	111.7	132.3	13.59	19.58	7.131	27.02	28.65	85.19
	$(15,13)$	25.32	70.19	25.59	38.82	1.019	2.567	0.368	1.468	1.688	16.70	188.0	245.4	189.7	211.8	13.59	19.58	2.595	39.04	39.05	217.1
	$(15,14)$	27.96	73.06	28.57	40.38	1.019	2.567	–	1.195	1.471	18.64†	204.0	261.3	204.5	226.0	13.59	19.58	–	37.37	40.03	244.7†



(a) Running Time (b) Communication Cost

Figure 9. The running time and communication cost of Case 1



(a) Running Time (b) Communication Cost

Figure 10. The running time and communication cost of Case 2

5.2. Comparison With Other Works

To answer question **Q2**, we compare our construction with the SOTA works [14][1] and [15][2]. The comparisons are made in the same environment (i.e., same machine and network setting) by running their source codes. Note that our construction relies on the offline/online paradigm. For maximum fairness, we review the code of [14] and [15] to remove operations that can be done offline in the benchmark.

1. https://github.com/private-panda/oring
2. https://github.com/orzcy/BZS-MPSI

In [14], the peeling phase of their OKVS encoding can be done offline. However, since it depends on each party's input set, it violates the constraints of our offline/online paradigm. In [15], the distribution of seeds in the OKVS sharing phase can be performed offline. We eliminate the running time and communication cost incurred by this operation in the following benchmark. Furthermore, both [14] and [15] rely on VOLE such that [14] leverages VOLE-based OPRF and [15] leverages VOLE-based two-party PSI [5]. However, directly measuring the cost of VOLE and eliminating it would lead to an incorrect result. For example, the Leader,

Table 3. THE COMMUNICATION COST COMPARISONS (IN MB) BETWEEN OUR PROTOCOL AND OTHER MPSI PROTOCOLS. THE BEST RESULT IS MARKED IN BLUE AND THE SECOND BEST RESULT IS MARKED IN ORANGE. CELLS WITH $-$ IMPLY THAT NO PARTIES ACT IN THIS ROLE IN THIS CASE. THE SIGN † IMPLIES THAT WE USE A TTP TO GENERATE THE CORRELATED RANDOMNESS FOR THE REDUCTION PHASE WHEN $t = n - 1$.

Protocol		Oring [14]			Kstar [14]			GLW [15]			Ours				
m	(n,t)	Client	Leader	Total	Client	Leader	Total	Client	Leader	Total	Client	Server	Pivot	Total (Online)	Total (Offline)
2^{12}	$(4,1)$	0.038	0.584	0.622	0.038	0.622	0.622	0.083	0.941	0.941	0.083	0.249	0.416	0.416	1.326
	$(4,2)$	1.055	2.109	3.442	1.055	2.147	3.442	0.083	0.941	0.941	0.083	0.249	0.582	0.582	3.977
	$(4,3)$	2.579	2.617	4.689	2.579	2.655	4.689	0.083	0.941	0.941	–	0.249	0.748	0.748	7.953†
	$(10,1)$	0.038	0.584	0.850	0.038	0.850	0.850	0.083	1.436	1.436	0.083	0.249	0.915	0.915	1.326
	$(10,4)$	2.071	6.176	18.44	2.071	6.441	18.44	0.083	1.436	1.436	0.083	0.249	1.413	1.413	13.26
	$(10,8)$	4.104	8.209	36.47	4.104	8.474	36.47	0.083	1.436	1.436	0.083	0.249	2.078	2.078	47.72
	$(10,9)$	8.679	8.717	41.51	8.679	8.983	41.51	0.083	1.436	1.436	–	0.249	2.244	2.244	59.65†
	$(15,1)$	0.038	0.584	1.041	0.038	1.040	1.041	0.083	1.849	1.849	0.083	0.249	1.330	1.330	1.326
	$(15,4)$	2.071	8.717	27.93	2.071	9.173	27.93	0.083	1.849	1.849	0.083	0.249	1.829	1.829	13.26
	$(15,7)$	3.193	10.24	48.21	3.596	10.70	48.21	0.083	1.849	1.849	0.083	0.249	2.328	2.328	37.12
	$(15,13)$	6.646	13.29	88.77	6.646	13.75	88.77	0.083	1.849	1.849	0.083	0.249	3.325	3.325	120.6
	$(15,14)$	13.76	13.80	100.2	13.76	14.26	100.2	0.083	1.849	1.849	–	0.249	3.491	3.491	139.2†
2^{16}	$(4,1)$	0.569	2.882	3.451	0.569	3.451	3.451	1.343	6.553	6.553	1.346	4.037	6.729	6.729	3.925
	$(4,2)$	4.055	8.110	13.61	4.055	8.679	13.61	1.343	6.553	6.553	1.346	4.037	9.420	9.420	11.78
	$(4,3)$	9.284	9.853	17.39	9.284	10.42	17.39	1.343	6.553	6.553	–	4.037	12.11	12.11	23.55†
	$(10,1)$	0.569	2.882	6.867	0.569	6.867	6.868	1.343	14.61	14.61	1.346	4.037	14.80	14.80	3.925
	$(10,4)$	7.541	22.05	70.88	7.541	26.04	70.88	1.343	14.61	14.61	1.346	4.037	22.88	22.88	39.25
	$(10,8)$	14.51	29.02	137.6	14.51	33.01	137.6	1.343	14.61	14.61	1.346	4.037	33.64	33.64	141.3
	$(10,9)$	30.20	30.77	146.3	30.20	34.75	146.3	1.343	14.61	14.61	–	4.037	36.33	36.33	176.6†
	$(15,1)$	0.569	2.882	9.715	0.569	9.714	9.715	1.343	21.33	21.33	1.346	4.037	21.53	21.53	3.926
	$(15,4)$	7.541	30.77	107.5	7.541	37.60	107.5	1.343	21.33	21.33	1.346	4.037	29.61	29.61	39.25
	$(15,7)$	12.77	36.00	182.6	12.77	42.83	182.6	1.343	21.33	21.33	1.346	4.037	37.68	37.68	109.9
	$(15,13)$	23.23	46.45	332.8	23.23	53.28	332.8	1.343	21.33	21.33	1.346	4.037	58.83	53.83	357.2
	$(15,14)$	47.63	48.20	349.6	47.63	55.03	349.6	1.343	21.33	21.33	–	4.037	56.52	56.52	412.1†
2^{20}	$(4,1)$	11.01	44.50	55.51	11.01	55.51	55.51	21.72	97.55	97.55	21.76	65.29	108.8	108.8	44.83
	$(4,2)$	55.98	112.0	190.0	55.98	123.0	190.0	21.72	97.55	97.55	21.76	65.29	152.3	152.3	134.5
	$(4,3)$	123.5	134.5	235.4	123.5	145.5	235.4	21.72	97.55	97.55	–	65.29	195.9	195.9	269.0†
	$(10,1)$	11.01	44.50	121.6	11.01	121.6	121.6	21.72	227.9	227.9	21.76	65.29	239.4	239.4	44.83
	$(10,4)$	101.0	291.9	973.1	101.0	368.9	973.1	21.72	227.9	227.9	21.76	65.29	369.9	369.9	448.3
	$(10,8)$	190.9	381.8	1869	190.9	458.9	1869	21.72	227.9	227.9	21.76	65.29	544.0	544.0	1614
	$(10,9)$	393.3	404.3	1921	393.3	481.4	1921	21.72	227.9	227.9	–	65.29	587.6	587.6	2017†
	$(15,1)$	11.01	44.50	176.6	11.01	176.6	176.6	21.72	336.5	336.5	21.76	65.29	348.2	348.2	44.83
	$(15,4)$	101.0	404.3	1476	101.0	536.4	1476	21.72	336.5	336.5	21.76	65.29	478.8	478.8	448.3
	$(15,7)$	168.4	471.8	2484	168.4	603.9	2484	21.72	336.5	336.5	21.76	65.29	609.3	609.3	1255
	$(15,13)$	303.4	606.7	4499	303.4	738.8	4499	21.72	336.5	336.5	21.76	65.29	870.5	870.5	4080
	$(15,14)$	618.2	629.2	4562	618.2	761.3	4562	21.72	336.5	336.5	–	65.29	914.0	914.0	4707†

who finally obtains the intersection, spends less running time than other parties. Overall, we only omit the distribution of seeds in the OKVS sharing phase and run their source code without any modification in the following benchmarks.

We evaluate these schemes on range of values $(n,t) \in \{(4,\{1,2,3\}),(10,\{1,4,8,9\}),(15,\{1,4,7,13,14\})\}$ on the set size $m \in \{2^{12},2^{16},2^{20}\}$.

The comparison result of running time is presented in Table 1. In the LAN setting, our construction achieves best running time. Compared with Oring and Kstar from [14], our construction is $9.57-114.46\times$ and $9.59-82.10\times$ faster in online running time, respectively. When considering a large set size, the total running time (offline+online) of our construction also outperforms Oring and Kstar. For example, with $m = 2^{20}$, the result shows that our construction is $4.20 - 18.73\times$ faster than Oring and $2.32 - 7.84\times$ faster than Kstar. Compared with GLW [15], our construction achieves comparable performance that is $1.52 - 26.03\times$ faster. We remark that GLW relies on the non-collusion assumption, while our construction resists arbitrary collusion. In the WAN setting, our construction still outperforms Oring and Kstar with $2.84 - 28.41\times$ and $2.69 - 28.00\times$ speedup, respectively. When the corruption threshold t is large, GLW outperforms our construction by up to $2.07\times$ in the case where $(n,t,m) = (15,14,2^{20})$. This is because our construction requires servers to engage in a secure $(t+1)$-party protocol to resist arbitrary collusion.

Furthermore, we present the comparison result of communication cost in Table 3. We observe that our construction has the best or the second-best communication cost, except when the corruption threshold is small. For a small set size $m = 2^{12}$, when n is small and t is large, our construction achieves the best performance. For example, for $(n,t) = (4,3)$, the online communication cost is 0.748 MB in total, which is 84.0% cheaper than Oring and Kstar, and 20.5% cheaper than GLW. When both the number of parties

and the corruption threshold are large, our construction generally achieves second-best performance. Specifically, for $(n, t, m) = (15, 14, 2^{20})$, our construction takes $2.72\times$ more communication cost than GLW but $4.99\times$ less cost than Oring and Kstar. Finally, existing works outperform our construction only in cases with a small corruption threshold (e.g., $t = 1$). In these cases, their construction has fewer communication rounds. Specifically, Oring, Kstar, and GLW completes in 2 rounds, while our construction requires 3 rounds: one each for reduction, mask, and reconstruction.

6. Conclusion

In this work, we focus on the mPSI protocol against arbitrary colluding parties of size $t < n$ in the semi-honest model. We abstract a new primitive called reducible zero-sharing. We provide a practical and secure construction by introducing an offline phase, which achieves linear total online complexity and resists arbitrary collusions against up to t colluding parties. Technically, we utilize the OKVS and VOLE. We fully leverage the homomorphic properties of OKVS to design a novel reducible zero-sharing construction. With reducible zero-sharing, our protocol outperforms the existing works with sufficient security and has similar performance, compared to works with security limitations.

In this paper, we only consider the semi-honest model and standard mPSI functionality. Our future work will focus on security improvement and functionality extension. Specifically, We believe using a random oracle is also helpful to provide malicious security as prior works [15], [18]. Additionally, the functionality of our reducible zero-sharing is currently simple. We can extend the functionality to support mPSI-variants. For example, when we let the output mapping hold the threshold property, we believe that a threshold reduction phase can lead to a threshold mPSI protocol. We keep these as future works.

7. Acknowledgments

We thank the shepherd and reviewers for helpful feedback on the write-up. This work was supported in part by the National Natural Science Foundation of China under Grant (No. 62572037), in part by Beijing Natural Science Foundation (No. 4242022), in part by the Open Research Fund of the State Key Laboratory of Blockchain and Data Security, Zhejiang University, and in part by PolyU (Project. No. P0044551).

References

[1] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious prf with applications to private set intersection," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 818–829.

[2] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "Spot-light: lightweight private set intersection from sparse ot extension," in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. Springer, 2019, pp. 401–431.

[3] ——, "Psi from paxos: Fast, malicious private set intersection," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2020, pp. 739–767.

[4] P. Rindal and P. Schoppmann, "Vole-psi: fast oprf and circuit-psi from vector-ole," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2021, pp. 901–930.

[5] S. Raghuraman and P. Rindal, "Blazing fast psi from improved okvs and subfield vole," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2505–2517.

[6] C. Hazay and M. Venkitasubramaniam, "Scalable multi-party private set-intersection," in *IACR international workshop on public key cryptography*. Springer, 2017, pp. 175–203.

[7] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu, "Practical multi-party private set intersection from symmetric-key techniques," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1257–1272.

[8] R. Inbar, E. Omri, and B. Pinkas, "Efficient scalable multiparty private set-intersection via garbled bloom filters," in *International conference on security and cryptography for networks*. Springer, 2018, pp. 235–252.

[9] A. Bay, Z. Erkin, M. Alishahi, and J. Vos, "Multi-party private set intersection protocols for practical applications," in *18th International Conference on Security and Cryptography, SECRYPT 2021*. SciTePress, 2021, pp. 515–522.

[10] N. Chandran, N. Dasgupta, D. Gupta, S. L. B. Obbattu, S. Sekar, and A. Shah, "Efficient linear multiparty psi and extensions to circuit/quorum psi," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1182–1204.

[11] O. Nevo, N. Trieu, and A. Yanai, "Simple, fast malicious multiparty private set intersection," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1151–1165.

[12] A. Bay, Z. Erkin, J.-H. Hoepman, S. Samardjiska, and J. Vos, "Practical multi-party private set intersection protocols," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1–15, 2021.

[13] J. Vos, M. Conti, and Z. Erkin, "Fast multi-party private set operations in the star topology from secure ands and ors," *Cryptology ePrint Archive*, 2022.

[14] M. Wu, T. H. Yuen, and K. Y. Chan, "O-ring and k-star: Efficient multi-party private set intersection," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 6489–6506.

[15] Y. Gao, Y. Luo, L. Wang, X. Liu, L. Qi, W. Wang, and M. Zhou, "Efficient scalable multi-party private set intersection (-variants) from bicentric zero-sharing," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 4137–4151.

[16] Y. Zhang, Y. Shi, Z. Zhou, C. Xue, Y. Xu, K. Xu, and J. Du, "Efficient and secure skyline queries over vertical data federation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 9, pp. 9269–9280, 2022.

[17] D. T. Nguyen and N. Trieu, "Mpccache: privacy-preserving multi-party cooperative cache sharing at the edge," in *International Conference on Financial Cryptography and Data Security*. Springer, 2022, pp. 80–99.

[18] G. Garimella, B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "Oblivious key-value stores and amplification for private set intersection," in *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II 41*. Springer, 2021, pp. 395–425.

[19] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2004, pp. 1–19.

[20] S. Ghosh and T. Nilges, "An algebraic approach to maliciously secure private set intersection," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2019, pp. 154–185.

[21] M. Dong, Y. Chen, C. Zhang, Y. Bai, and Y. Cao, "Multi-party private set operations from predicative zero-sharing," *arXiv preprint arXiv:2504.20050*, 2025.

[22] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2001, vol. 2.

[23] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai, "Compressing vector ole," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 896–912.

[24] S. Raghuraman, P. Rindal, and T. Tanguy, "Expand-convolute codes for pseudorandom correlation generators from lpn," in *Annual International Cryptology Conference*. Springer, 2023, pp. 602–632.

[25] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," in *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, 1990, pp. 503–513.

[26] L. R. Peter Rindal, "libOTe: an efficient, portable, and easy to use Oblivious Transfer Library," https://github.com/osu-crypto/libOTe.

[27] Visa-Research, "Vole-PSI," https://github.com/Visa-Research/volepsi.

Appendix A. Security Proof of Theorem 1

Let $\mathcal{C}$ and $\mathcal{H}$ be a coalition of corrupted and honest parties, respectively. We first prove that the triple is indistinguishable from a random triple. We consider $\Pi^{n,m}_{\mathsf{tg}}$. The simulator Sim needs $\mathcal{C}$'s views. Sim runs as follows:

- For step $1), 4), 5)$, Sim runs the protocol honestly.
- For step $2)$, without loss of generality, Sim simulates P_i's view by uniformly distributing $\vec{C}'_{ij}, \vec{A}'_{ji}, \vec{B}'_{ji} \in \mathbb{F}^m$ for $j \in [1, n] \setminus \{i\}$ and appending these vectors to P_i's view.
- For step $3)$, Sim simulates P_j's view by sending a random $\vec{A}^*_{ij} \in \mathbb{F}^m$ to P_i.

We are going to show that the simulated execution is indistinguishable from the real protocol execution.

Hybrid $\mathcal{H}_0$**.** Same as real-world execution in $\mathcal{F}^{\mathbb{F},m}_{\mathsf{vole}}$-hybrid.

Hybrid $\mathcal{H}_1$**.** This hybrid is identical to $\mathcal{H}_0$ except Sim emulates step $2)$. According to the security of $\mathcal{F}^{\mathbb{F},m}_{\mathsf{vole}}$, the vectors obtained in step $2)$ are computationally indistinguishable from random vectors. Thus, $\mathcal{H}_1$ is indistinguishable from $\mathcal{H}_0$.

Hybrid $\mathcal{H}_2$**.** This hybrid is identical to $\mathcal{H}_1$ except Sim emulates step $3)$. Since $\vec{A}'_{ij}$ is computationally indistinguishable from a random vector, computing $\vec{A}^*_{ij}$ is a one-time pad encryption. Thus, $\vec{A}^*_{ij}$ is also computationally indistinguishable from a random vector. Therefore, $\mathcal{H}_2$ is indistinguishable from $\mathcal{H}_1$.

Overall, the view simulated by Sim is identical to hybrid $\mathcal{H}_0$, and the security holds.

We then prove that with the triple, the output of the mRSM protocol is indistinguishable from a random vector. We consider $\Pi^{n,m}_{\mathsf{mrsm}}$.

Let $\mathcal{C}$ and $\mathcal{H}$ be a coalition of corrupted and honest parties, respectively. The simulator Sim needs $\mathcal{C}$'s views. Sim runs as follows:

- For step $1)$, Sim simulates views by sending a random $\vec{X}_i \in \mathbb{F}^m$ to P_n.
- For step $2)$, Sim simulates views by sending a random $\vec{X}' \in \mathbb{F}^m$ to P_i.
- For step $3)$, Sim runs the protocol honestly.

We are going to show that the simulated execution is indistinguishable from the real protocol execution.

Hybrid $\mathcal{H}_0$**.** Same as real-world execution.

Hybrid $\mathcal{H}_1$**.** This hybrid is identical to $\mathcal{H}_0$ except Sim emulates step $1)$. Since $\vec{A}_i$ is computationally indistinguishable from a random vector, computing $\vec{X}'_i$ is a one-time pad encryption. Thus, $\vec{X}'_i$ is also computationally indistinguishable from a random vector. Therefore, $\mathcal{H}_1$ is indistinguishable from $\mathcal{H}_0$.

Hybrid $\mathcal{H}_2$**.** This hybrid is identical to $\mathcal{H}_1$ except Sim emulates step $2)$. Since all $\vec{X}'_i$ are computationally indistinguishable from random vectors, $\vec{X}$ is computationally indistinguishable from a random vector. Therefore, $\mathcal{H}_2$ is indistinguishable from $\mathcal{H}_1$.

Overall, the view simulated by Sim is identical to hybrid $\mathcal{H}_0$, and the security holds.

Finally, we prove Theorem 1. Let $\mathcal{C}$ and $\mathcal{H}$ be a coalition of corrupted and honest parties, respectively. Since the security of the offline phase is ensured by PRG and $\mathcal{F}^{n,m}_{tg}$, we only prove the security of the online phase. We divide the proof into two cases, depending on whether the pivot P_n is corrupted.

P_n ***is corrupted.*** Sim first invokes the simulator of $\mathcal{F}^{n,m}_{\mathsf{mrsm}}$ to obtain the output $\vec{M}_n$, appending it to the view of P_n. After receiving the $\{f_i | i \in \mathcal{C}\}$ from the ideal world, Sim generates t sets of key-value pairs $P_j = \{(x, s^i)\}$ for $i \in \mathcal{C}$ such that $\bigoplus_{i\in\mathcal{C}} s^i = \bigoplus_{i\in\mathcal{C}} r\mathsf{Decode}(f_i, x)$, where r is a random value. Then, Sim encodes OKVS objects $\vec{M}_j = \mathsf{Encode}(P_j)$, and appends these objects to the view of P_n. Finally, Sim uniformly samples $n-t$ OKVS objects and appends them to the view of P_n.

Assume that some server P_j is corrupted in this case, Sim appends M_j and the output of the simulator of $\mathcal{F}^{n,m}_{\mathsf{mrsm}}$ to the view of P_j.

Assume that some client P_i is corrupted in this case, its view can be directly simulated by uniformly sampling an OKVS object.

Therefore, the simulated view output by Sim is indistinguishable from the real.

P_n ***is honest.*** Sim is only required to simulate the view of client and server in this case. To simulate the view of client, Sim uniformly distributes an OKVS object and appends it to the view of client. To simulate the view of server, Sim first appends a random OKVS object to the view. Then, Sim invokes the simulator of $\mathcal{F}^{n,m}_{\mathsf{mrsm}}$ and appends the output to the view.

Therefore, the simulated view output by Sim is indistinguishable from the real.

Appendix B.
Security Proof of Theorem 2

Let $\mathcal{C}$ and $\mathcal{H}$ be a coalition of corrupted and honest parties, respectively. The simulator Sim needs $\mathcal{C}$'s views. Sim runs as follows:

- For step 1), Sim invokes the reducible zero-sharing simulator $\mathsf{Sim}_{\mathsf{rzs}}^{n,t,m}$ and appends the output to the view.
- For step 2), Sim generates random OKVS objects and appends the output to the view.

Since the output of $\mathsf{Sim}_{\mathsf{rzs}}^{n,t,m}$ is indistinguishable from a random vector and the servers perform a one-time pad, the simulated view by Sim is indistinguishable from the real.

Appendix C.
Performance Comparison with [21]

We have conducted additional experiments to compare our protocol with [21]. Since [21] is not open-sourced, we rely on the results reported in their paper. To ensure fairness, we adopt their network settings, that is 10Gbps bandwidth and 0.1ms RTT latency for the LAN setting and 400Mbps and 80ms RTT latency for the WAN setting. We consider set sizes $m \in \{2^{12}, 2^{16}, 2^{20}\}$ reported in their paper. For the number of parties $n \in \{4, 10\}$, we set the corruption threshold $t \in \{n-1, n-2\}$ to match the worst-case costs while the costs of [21] do not depend on t. Since [21] does not report the offline phase result, we compare only the online phase.

The result is reported in Table 4. In the LAN setting, our protocol achieves $3.67 - 8.67\times$ speedup, while it is $2.34 - 5.20\times$ in the WAN setting. In addition, our protocol also reduces the communication cost by $1.25 - 1.65\times$.

Table 4. THE ONLINE RUNNING TIME (IN SECONDS) AND COMMUNICATION COST (IN MB) COMPARISON OF [21] AND OUR PROTOCOL. THE BEST RESULT IS MARKED IN BLUE.

Settings		Online running time				Communication cost	
		LAN		WAN			
m	(n,t)	[21]	Ours	[21]	Ours	[21]	Ours
2^{12}	$(4,2)$	0.015	0.004	1.944	0.374	0.961	0.582
	$(4,3)$	0.015	0.003	1.944	0.216	0.961	0.748
	$(10,8)$	0.026	0.004	4.363	0.393	2.884	2.078
	$(10,9)$	0.026	0.003	4.363	0.233	2.884	2.244
2^{16}	$(4,2)$	0.242	0.066	3.399	1.052	15.31	9.420
	$(4,3)$	0.242	0.047	3.399	0.706	15.31	12.11
	$(10,8)$	0.354	0.087	5.769	2.090	45.92	33.64
	$(10,9)$	0.354	0.077	5.769	1.794	45.92	36.33
2^{20}	$(4,2)$	8.133	0.842	19.23	5.852	246.2	152.3
	$(4,3)$	8.133	0.658	19.23	5.698	246.2	195.9
	$(10,8)$	8.898	1.319	33.38	14.30	738.7	544.0
	$(10,9)$	8.898	1.107	33.38	14.26	738.7	587.6