Maliciously Secure Multi-Party PSI with Lower Bandwidth and Faster Computation

Zhi Qiu
Shanghai Jiao Tong University
chonps@sjtu.edu.cn

Kang Yang
State Key Laboratory of Cryptology
yangk@sklc.org

Yu Yu
Shanghai Jiao Tong University
yuyuathk@gmail.com

Lijing Zhou
Huawei Technology
zhoulijing@huawei.com

June 15, 2022

Abstract

Private Set Intersection (PSI) allows a set of mutually distrustful parties, each holds a private data set, to compute the intersection of all sets, such that no information is revealed except for the intersection. The state-of-the-art PSI protocol (Garimella et al., CRYPTO'21) in the multi-party setting tolerating any number of malicious corruptions requires the communication bandwidth of $O(n\ell|\mathbb{F}|)$ bits for the central party P_0 due to the star architecture, where n is the number of parties, ℓ is the size of each set and $|\mathbb{F}|$ is the size of an exponentially large field $\mathbb{F}$. When n and ℓ are large, this forms an efficiency bottleneck (especially for networks with restricted bandwidthes). In this paper, we present a new multi-party PSI protocol in dishonest-majority malicious setting, which reduces the communication bandwidth of the central party P_0 from $O(n\ell|\mathbb{F}|)$ bits to $O(\ell|\mathbb{F}|)$ bits using a tree architecture. Furthermore, our PSI protocol reduces the expensive LPN encoding operations performed by P_0 by a factor of n as well as the computational cost by $2n\ell$ hash operations in total. Additionally, while the multi-party PSI protocol (Garimella et al., CRYPTO'21) with a single output is secure, we present a simple attack against its multi-output extension, which allows an adversary to learn more information on the sets of honest parties beyond the intersection of all sets.

1 Introduction

Private Set Intersection (PSI) allows a set of mutually distrustful parties, where each holds a private set, to compute the intersection of all sets, without revealing anything beyond the intersection. PSI and its variants have found a wide variety of applications, including measuring the effectiveness of online advertising [IKN+17, IKN+20], private contact discovery [KRS+19, DRRT18] and more. In the two-party setting, PSI protocols has been extensively studied and become truly practical with extremely fast implementations (see the recent work [GN19, PRTY19, PRTY20, CM20, RS21, GPR+21, RR22] and references therein). While two-party PSI is interesting for many applications, there are a lot of applications which are better suitable for the multi-party setting. For example, a) several companies intend to combine their data sets to find a target audience for an ad campaign [IKN+20]; b) a variant of multi-party PSI was recently used for cache sharing in edge computing, which allows multiple network operators to obtain a set of common data items with the highest access frequencies in the privacy-preserving way [NT21]. We refer the reader to [NTY21] for more interesting examples that are suited to the multi-party case.

The problem of multi-party PSI was first introduced in [FNP04]. The previous work [FNP04, KS05, SS07, SS08, CJS10, HV17, GN19] constructed theoretical multi-party PSI protocols, where all of these protocols are not concretely efficient (especially for large sets). The first practical PSI protocol in the multi-party setting was proposed by Kolesnikov et al. [KMP+17]. This protocol is secure against semi-honest adversaries in the dishonest-majority setting (i.e., the adversary can corrupt up to $n-1$ parties of the n parties but must follow the protocol specification). For semi-honest security, efficient multi-party PSI protocols were further developed, including [IOP18, KMS21] in the dishonest-majority setting based on Garbled Bloom Filter (GBF), and [CDG+21] in the honest-majority setting (i.e., the adversary can corrupt up to less than a half of parties). In this work, we focus on multi-party PSI protocols in the dishonest-majority setting in the presence of malicious adversaries (i.e, the adversary allows to run an arbitrary attack strategy in its attempt to break the protocol). This is the strongest adversary model that was considered in the previous PSI protocols.

In the dishonest-majority malicious setting, several concretely efficient PSI protocols [ZLL+19, ENOPC21, GPR+21, NTY21] have been proposed. Among these protocols, the multi-party PSI protocol by Garimella et al. [GPR+21] achieves the best efficiency in this setting. This protocol builds on the multi-party PSI protocol with augmented semi-honest security [KMP+17], and instantiates the underlying Oblivious Programmable Pseudo-Random Function (OPPRF) primitive with the Oblivious Key-Value Store (OKVS) scheme [GPR+21] and Oblivious Pseudo-Random Function (OPRF). Garimella et al. [GPR+21] use a 3-hash garbled cuckoo table to design the OKVS scheme, which is an optimized version of the PaXoS construction [PRTY20] and can achieve much better communication efficiency than GBF. The state-of-the-art protocol [RS21] to realize OPRF adopts the recent LPN-based Vector Oblivious Linear-function Evaluation (VOLE) protocol with *sublinear* communication [BCGI18, BCG+19b, BCG+19a, SGRR19, BCG+20, WYKW21, CRR21]. Garimella et al. [GPR+21] modified the augmented semi-honest PSI protocol [KMP+17] into a maliciously secure protocol by adding a random oracle to wrap the OPRF output, [1] and proved that the modified multi-party PSI protocol is secure against malicious adversaries. Although the state-of-the-art multi-party PSI protocol [GPR+21] is concretely efficient, we observed the following two aspects that need to be further improved and are addressed by this work.

- The multi-party PSI protocol [GPR+21] adopts the *star* architecture for communication. Specifically, the central party P_0 will interact with parties $P_1, \ldots, P_n$ to compute n OPRF outputs that will include n VOLE protocol executions, and then receives n OKVS from the n parties. When n and the size ℓ of each set are large, the communication bandwidth of P_0 is $O(n\ell|\mathbb{F}|)$ bits, which forms an efficiency bottleneck of the protocol (especially for networks with restricted bandwidthes), where $|\mathbb{F}|$ is the size of an exponentially large field $\mathbb{F}$.

- The n VOLE executions for computing n OPRF outputs will require n encoding operations of Learning Parity with Noise (LPN) for the central party P_0, where the LPN encoding is computationally expensive [YWL+20] and forms a computational efficiency bottleneck of the state-of-the-art VOLE protocol with malicious security [WYKW21].

1.1 Our Contributions

In this paper, we propose a new multi-party PSI protocol in the dishonest-majority malicious setting, which improves the state-of-the-art multi-party PSI protocol [GPR+21] in the following two aspects.

[1] A similar observation was also made by Nevo et al. [NTY21].

MP-PSI	Topology	Communication	Bandwidth	Rounds
[ENOPC21]	Star	$O(n\ell\kappa^2 + n\ell\kappa\log(\ell\kappa))$	$O(n\ell\kappa^2 + n\ell\kappa\log(\ell\kappa))$	4
[GPR+21]	Star	$O(n^2\kappa + n\ell(\kappa+\rho+\log\ell))$	$O(n\ell(\kappa+\rho+\log\ell))$	2
This work	Tree	$O(n^2\kappa + n\ell(\kappa+\rho+\log\ell))$	$O(\ell(\kappa+\rho+\log\ell))$	$2\lfloor\log(n+1)\rfloor$

Table 1: **Comparison between our protocol and recent multi-party PSI protocols tolerating all-but-one malicious corruptions.** $n+1$ is the number of parties, ℓ is the size of each set, and κ and ρ are the computational and statistical security parameters respectively. The bandwidth column denotes the maximum communication sent or received by each party.

- We reduce the computation cost of LPN encoding for the central party P_0 implied in the PSI protocol [GPR+21] by a factor of n. Meanwhile, we further reduce the total computation cost of their PSI protocol by $2n\ell$ hash operations. To achieve the efficiency gain, we construct the PSI protocol by directly using VOLE in the multi-party setting (instead of calling OPRF) and integrating the repetitive operations (see Section 3.2 for a technical overview).

- Building on the above technique to improve computation, we use a *tree* architecture to reduce the communication bandwidth of the central party P_0. In particular, we reduce the bandwidth complexity of P_0 from $O(n\ell)$ field elements to $O(\ell)$ field elements by amortizing the communication among all parties in a tree network architecture. We present two types of tree architectures: one is used to make P_0 send the same message to parties $P_1, \ldots, P_n$; and the other is used to let $P_1, \ldots, P_n$ send the sum of n different messages to P_0 in an aggregation way (see Section 3.3 for a technical overview).

Our PSI protocol requires the underlying OKVS scheme to be *linear*, which is satisfied by the recent efficient constructions [PRTY20, GPR+21, RR22]. We prove the security of our protocol in the Universal Composability (UC) model and the random oracle model.

In addition, Garimella et al. [GPR+21] proposed a multi-output extension of their multi-party PSI protocol such that all parties can obtain the output instead of only the central party P_0. While their multi-party PSI protocol with a single output is provably secure, we present a simple attack against the multi-output extension. This attack allows an adversary to leak the information of the sets held by honest parties more than that allowing to be obtained from the intersection of the sets of all parties, even if the adversary behaves semi-honestly. In particular, if only P_0 is honest, then the adversary can leak some secret items in the private set of P_0.

Comparison of communication, bandwidth and rounds. In Table 1, we compare our protocol with two recent multi-party PSI protocols in the dishonest-majority malicious setting (where the adversary can corrupt up to n parties of the $n+1$ parties). For the sake of simplicity, this table does not compare the recent PSI protocol [NTY21], as it has the same complexities as [GPR+21] in this setting. All costs are counted in the ROT/VOLE-hybrid model, where ROT represents random oblivious transfer. For the state-of-the-art ROT or VOLE protocols, the communication of these protocols is small compared to the whole communication. From this table, we can see that our protocol has the lowest bandwidth complexity while keeping the same communication complexity. In particular, our tree architecture does not increase the total communication cost, compared to the state-of-the-art multi-party PSI protocol [GPR+21]. As a trade-off of lower communication bandwidth, the round complexity of our PSI protocol is increased to $O(\log n)$, compared to the multi-party PSI protocol [GPR+21, NTY21] with the round complexity of $O(1)$.

Functionality $\mathcal{F}_{\mathsf{mpsi}}$

Parameters: The size ℓ of the input set for every honest party. The maximal allowed size $\ell' \geq \ell$ of the input set for any malicious party. Let $\mathcal{C}$ be the set of corrupted parties.

Execution: This functionality runs with parties $P_0, \ldots, P_n$. Upon receiving (input, i, X_i) from every party P_i for $i \in [0, n]$, this functionality executes as follows:

1. If there exists some $i \in \mathcal{C}$ such that $|X_i| > \ell'$, send abort to all parties and then abort.
2. If there exists some $i \notin \mathcal{C}$ such that $|X_i| > \ell$, send abort to all parties and then abort.
3. Otherwise, compute $Y := \bigcap_{i \in [0,n]} X_i$, and then send Y to P_0.

Figure 1: **Functionality for multi-party private set intersection.**

2 Preliminaries

2.1 Notation

We use κ and λ to denote the computational and statistical security parameters, respectively. For $a, b \in \mathbb{N}$ and $a \leq b$, we use $[a, b]$ to denote the set $\{a, \ldots, b\}$. For a finite set S, we use $x \leftarrow S$ to denote sampling x uniformly at random from S. For a vector $\boldsymbol{x}$, we denote by x_i the i-th component of $\boldsymbol{x}$ with x_1 the first entry. For a set S, we use $|S|$ to denote the size of set S.

2.2 Security Model and Functionalities

Security model. We use the Universal Composability (UC) framework [Can01] to prove security in the presence of a static, malicious adversary. We say that a protocol Π *UC-realizes* an ideal functionality $\mathcal{F}$ if for any Probabilistic Polynomial Time (PPT) adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$, such that for any PPT environment $\mathcal{Z}$, the output distribution of $\mathcal{Z}$ in the *real-world* execution where the parties interact with $\mathcal{A}$ and execute Π is computationally indistinguishable from the output distribution of $\mathcal{Z}$ in the *ideal-world* execution where the parties interact with $\mathcal{S}$ and $\mathcal{F}$.

Functionality for multi-party PSI. In a PSI protocol, $n+1$ parties $P_0, P_1, \ldots, P_n$ with each having an input set X_i of size ℓ compute the intersection of their input sets, i.e., $\bigcap_{i \in [0,n]} X_i$. As a result of the protocol execution, P_0 obtains the intersection output, and all other parties learn nothing. The multi-party PSI functionality is shown in Figure 1. Following prior work such as [RR17, PRTY20, RS21, ENOPC21], we allow the adversary who corrupts a party P_i to input a set X_i with $\ell \leq |X_i| \leq \ell'$.

Functionality for VOLE. Following the previous definition [BCG+19a, WYKW21], the functionality for Vector Oblivious Linear-function Evaluation (VOLE) is given in Figure 2. Two parties run the initialization procedure only once, and then can repeatedly call the extend procedure to obtain multiple batches of VOLE correlations. The VOLE functionality can be securely realized by the recent LPN-based protocols with *sublinear* communication [BCGI18, BCG+19b, BCG+19a, SGRR19, BCG+20, WYKW21, CRR21].

2.3 Oblivious Key-Value Stores

We recall the definitions of an Oblivious Key-Value Store (OKVS) proposed by Garimella et al. [GPR+21]. OKVS is a general notion, and captures the functionality and security property

Functionality $\mathcal{F}_{\mathsf{vole}}$

This functionality runs with a sender P_S, a receiver P_R and an adversary, and operates as follows:

Initialize: Upon receiving (init) from P_S and P_R, if P_S is honest, then sample $\Delta \leftarrow \mathbb{F}$, otherwise receive $\Delta \in \mathbb{F}$ from the adversary. Store Δ and output Δ to P_S, and ignore all subsequent (init) commands.

Extend: Upon receiving (extend, m) from P_S and P_R, do the following:

1. If P_S is honest, sample $\boldsymbol{v} \leftarrow \mathbb{F}^m$. Otherwise, receive $\boldsymbol{v} \in \mathbb{F}^m$ from the adversary.
2. If P_R is honest, sample $\boldsymbol{u} \leftarrow \mathbb{F}^m$ and compute $\boldsymbol{w} := \boldsymbol{v} + \boldsymbol{u} \cdot \Delta$. Otherwise, receive $\boldsymbol{u}, \boldsymbol{w} \in \mathbb{F}^m$ from the adversary, and then recompute $\boldsymbol{v} := \boldsymbol{w} - \boldsymbol{u} \cdot \Delta$.
3. Output $\boldsymbol{v}$ to P_S and $(\boldsymbol{u}, \boldsymbol{w})$ to P_R.

Figure 2: **Functionality for vector oblivious linear-function evaluation.**

of existing constructions, including polynomials, dense matrix, garbled Bloom filter [DCW13] and PaXoS [PRTY20]. Then, we roughly discuss the state-of-the-art OKVS construction as well as its communication and computation complexities.

Definition 1 ([GPR+21])**.** *A key-value store is parameterized by a set $\mathcal{K}$ of keys, a set $\mathcal{V}$ of values, and a set $\mathcal{H}$ of functions, and consists of the following two algorithms:*

- $\mathsf{Encode}_{\mathcal{H}}(\{(k_i, v_i)\}_{i\in[1,\ell]})$ *takes as input a set of key-value pairs $\{(k_i, v_i)\}_{i\in[1,\ell]}$, and outputs an object $\mathbf{S}$ (or an error indicator $\perp$ with statistically small probability).*
- $\mathsf{Decode}_{\mathcal{H}}(\mathbf{S}, k)$ *takes as input an object $\mathbf{S}$, a key k, and outputs a value v.*

A KVS is correct, if for all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys:

$$(k, v) \in A \text{ and } \perp \neq \mathbf{S} \leftarrow \mathsf{Encode}_{\mathcal{H}}(A) \Rightarrow \mathsf{Decode}_{\mathcal{H}}(\mathbf{S}, k) = v.$$

For the sake of simplicity, we choose to omit the underlying parameter $\mathcal{H}$ in the rest of exposition, as long as the context is clear. In the known OKVS constructions, we always have that the decision whether Encode outputs $\perp$ depends on the functions $\mathcal{H}$ and the keys $\{k_i\}_{i\in[\ell]}$, and is independent of the values $\{v_i\}_{i\in[\ell]}$. In the following, we define the security property guaranteeing that one cannot decide whether a key k was used to generate $\mathbf{S}$ or not.

Definition 2 ([GPR+21])**.** *A KVS is an Oblivious KVS (OKVS), if for all distinct keys $\{k_1^0, ..., k_\ell^0\}$ and all distinct keys $\{k_1^1, ..., k_\ell^1\}$, when Encode does not output $\perp$ for both $(k_1^0, ..., k_\ell^0)$ and $(k_1^1, ..., k_\ell^1)$, the output of $\mathcal{R}(k_1^0, ..., k_\ell^0)$ is computationally indistinguishable from that of $\mathcal{R}(k_1^1, ..., k_\ell^1)$, where $\mathcal{R}(k_1, ..., k_\ell)$ is defined as:*

- *For $i \in [1, \ell]$, sample $v_i \leftarrow \mathcal{V}$.*
- *Output $\mathsf{Encode}(\{(k_i, v_i)\}_{i\in[1,\ell]})$.*

From the above definition, we have that if the OKVS encodes random values, for any two sets of keys K^0, K^1, it is infeasible to distinguish the OKVS encoding of the keys of K^0 from that of the keys of K^1.

Our multi-party PSI protocol requires that an OKVS has some kind of additively homomorphic property. In particular, $\mathsf{Decode}(\cdot, k)$ is a *linear function* for all $k \in \mathcal{K}$. The formal definition is recalled as follows.

Definition 3 ([GPR+21]). *An OKVS is linear over a field $\mathbb{F}$ if $\mathcal{V} = \mathbb{F}$ (i.e., "values" are elements of $\mathbb{F}$), then the output of* Encode *is a vector* $\mathbf{S}$ *in* $\mathbb{F}^m$*, and the* Decode *function is defined as:*

$$\mathsf{Decode}(\mathbf{S}, k) = \langle d(k), \mathbf{S} \rangle \stackrel{\text{def}}{=} \sum_{i=1}^{m} d(k)_i \cdot S_i$$

for some function $d : \mathcal{K} \to \mathbb{F}^m$, where d is typically defined by hash functions. Thus, $\mathsf{Decode}(\cdot, k)$ *is a linear map from $\mathbb{F}^m$ to $\mathbb{F}$.*

The idea of constructing a linear OKVS is that generating a solution to the linear system of equations:

$$\begin{bmatrix} -\,d(k_1)\,- \\ -\,d(k_2)\,- \\ \vdots \\ -\,d(k_\ell)\,- \end{bmatrix} \cdot \mathbf{S}^\top = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_\ell \end{bmatrix}.$$

If Encode chooses uniformly from the set of solutions to the linear system and the values are uniform, then the output $\mathbf{S}$ is uniformly distributed (and thus independent of the keys). That is, a linear OKVS satisfies the obliviousness property. The recent linear OKVS scheme [GPR+21] (building on the PaXoS technique [PRTY20]) has the computation complexity *linear* to the number of key-value pairs, and achieves the rate of $0.81 - o(1)$, where the rate of an OKVS that encodes ℓ items from $\mathbb{F}$ is the ratio between $\ell \cdot |\mathbb{F}|$ and the size of the OKVS. Very recently, Rindal and Raghuraman [RR22] significantly improved the computation efficiency of the linear OKVS scheme [GPR+21], and achieve the best concrete performance for now.

An OKVS whose parameters are chosen to encode N items may hold even more than N items, when it is generated by the adversary. In the context of PSI, this allows the adversary to encode more items than advertised. In Appendix A, we review the the definition of OKVS overfitting to bound the number of items that the adversary can "overfit" into an OKVS, which will be used in the security proof of our PSI protocol.

3 Technical Overview

In this section, we give an overview of our techniques to improve the communication bandwidth and computation cost of the state-of-the-art multi-party PSI protocol [GPR+21] tolerating any number of malicious corruptions. The PSI protocol by Garimella et al. [GPR+21] is constructed by transforming the augmented semi-honest PSI protocol [KMP+17] into a maliciously secure version by adding a random oracle to wrap the OPRF output, which is also observed by Nevo et al. [NTY21]. Firstly, we review the multi-party PSI protocol [GPR+21] at a high level.

3.1 Overview of the Best-Known Multi-Party PSI Protocol

The state-of-the-art multi-party PSI protocol [GPR+21] tolerating any number of malicious corruptions, which builds on the augmented semi-honest protocol [KMP+17], executes as follows:

1. $P_0, P_1, \ldots P_n$ are the $n+1$ parties who will compute the intersection of input sets $X_0, X_1, \ldots, X_n$ with $|X_i| = \ell$ for $i \in [0, n]$, where P_0 will obtain the output. For any h, every party P_i can compute a zero share s_h^i such that $\sum_{i \in [0,n]} s_h^i = 0$ by exchanging Pseudo-Random Function (PRF) keys.

Functionality $\mathcal{F}_{\mathsf{mvole}}$

Let $\mathcal{C}$ be the set of corrupted parties. This functionality runs with parties $P_0, P_1, \ldots, P_n$ and an adversary, and operates as follows:

Initialize: For each $i \in [1, n]$, upon receiving (init, i) from P_i, if P_i is honest, then sample $\Delta_i \leftarrow \mathbb{F}$, else receive $\Delta_i \in \mathbb{F}$ from the adversary. Store $(\Delta_1, \ldots, \Delta_n)$ and output Δ_i to P_i for each $i \in [1, n]$, and ignore all subsequent (init) commands.

Extend: Upon receiving (extend, m) from all parties, execute as follows:

1. For $i \in [1, n]$, if P_i is honest, then sample $\boldsymbol{v}_i \leftarrow \mathbb{F}^m$, else receive $\boldsymbol{v}_i \in \mathbb{F}^m$ from the adversary.
2. If P_0 is honest, sample $\boldsymbol{u} \leftarrow \mathbb{F}^m$ and compute $\boldsymbol{w}_i := \boldsymbol{v}_i + \boldsymbol{u} \cdot \Delta_i$ for $i \in [1, n]$.
3. If P_0 is corrupted, receive $\boldsymbol{u}_i \in \mathbb{F}^m$ for all $i \in [1, n]$ and $\boldsymbol{w}_i \in \mathbb{F}^m$ for all $i \notin \mathcal{C}$ from the adversary, and then recompute $\boldsymbol{v}_i := \boldsymbol{w}_i - \boldsymbol{u}_i \cdot \Delta_i$ for each $i \notin \mathcal{C}$ and compute $\boldsymbol{w}_i := \boldsymbol{v}_i + \boldsymbol{u}_i \cdot \Delta_i$ for each $i \in \mathcal{C}$.
4. Output $\{(\boldsymbol{u}_i, \boldsymbol{w}_i)\}_{i \in [1,n]}$ to P_0 and $\boldsymbol{v}_i$ to P_i for $i \in [1, n]$.

Figure 3: **Functionality for multi-party VOLE.**

2. For each $i \in [1, n]$, P_0 and P_i call an oblivious PRF functionality $\mathcal{F}_{\mathsf{oprf}}$ with malicious security, and then P_i obtains the key denoted by PRF_i and P_0 gets the set $\{\mathsf{PRF}_i(h) \,|\, h \in X_0\}$.

3. Let $(\mathsf{Encode}, \mathsf{Decode})$ be an OKVS scheme which maps ℓ items to m slots, and $\mathsf{H} : \{0,1\}^* \to \mathbb{F}$ be a random oracle. For each $i \in [1, n]$, P_i computes an OKVS $\mathbf{Q}_i$ as

$$\mathbf{Q}_i = \mathsf{Encode}\left(\left\{(h, \mathsf{H}(\mathsf{PRF}_i(h), h) + s_h^i) \,|\, h \in X_i\right\}\right),$$

   and then sends $\mathbf{Q}_i$ to P_0.

4. P_0 computes an OKVS $\mathbf{Q}_0$ as

$$\mathbf{Q}_0 = \mathsf{Encode}\Big(\Big\{(h, -\sum_{i=1}^{n} \mathsf{H}(\mathsf{PRF}_i(h), h) + s_h^0) \,\Big|\, h \in X_0\Big\}\Big).$$

5. After receiving the OKVS from all other n parties, P_0 computes the output as

$$\Big\{h \in X_0 \,\Big|\, \sum_{i=0}^{n} \mathsf{Decode}(\mathbf{Q}_i, h) = 0\Big\}.$$

3.2 Our Approach to Improve Computation Efficiency

The state-of-the-art OPRF protocol [RS21] is constructed by combining VOLE with an OKVS scheme called PaXoS [PRTY20], where the efficiency of OKVS can be further improved using the recent constructions [GPR+21, RR22]. At a high level, this protocol running between P_i and P_0 works as follows:

1. P_i and P_0 call functionality $\mathcal{F}_{\mathsf{vole}}$ (shown in Figure 2), which returns $\boldsymbol{v}_i$ to P_i and $(\boldsymbol{u}, \boldsymbol{w}_i)$ to P_0, where $\boldsymbol{w}_i = \boldsymbol{v}_i + \boldsymbol{u} \cdot \Delta_i$.

2. For an input set X_0, P_0 computes an OKVS as:

$$\mathbf{S} = \mathsf{Encode}\left(\{(h, \mathsf{H}(h)) \,|\, h \in X_0\}\right).$$

3. P_0 computes $\boldsymbol{d} := \mathbf{S} - \boldsymbol{u} \in \mathbb{F}^m$ and sends it to P_i who computes $\mathbf{V}_i := \boldsymbol{v}_i - \boldsymbol{d} \cdot \Delta_i$. Let $\mathbf{W}_i = \boldsymbol{w}_i$, and thus $\mathbf{W}_i = \mathbf{V}_i + \mathbf{S} \cdot \Delta_i$.

4. P_0 computes $\mathsf{PRF}_i(h) = \mathsf{H}(\mathsf{Decode}(\mathbf{W}_i, h), h)$ for $h \in X_0$, and P_i can compute $\mathsf{PRF}_i(x) = \mathsf{H}(\mathsf{Decode}(\mathbf{V}_i, x) + \mathsf{H}(x) \cdot \Delta_i, x)$ for any x in the domain.

If integrating the above OPRF protocol into the state-of-the-art multi-party PSI protocol with malicious security shown in the previous section, P_0 needs to call $\mathcal{F}_{\mathsf{vole}}$ with n different parties. Note that the LPN encoding is the computational efficiency bottleneck for the state-of-the-art VOLE protocol [WYKW21] in the malicious setting. When instantiating functionality $\mathcal{F}_{\mathsf{vole}}$, the multi-party PSI protocol requires P_0 to perform n operations of LPN encoding, which is computationally expensive.

Our solution. Our approach reduces the cost of LPN encoding associated with vector $\boldsymbol{u}$ by a factor of n. We make an important observation that a single random vector $\boldsymbol{u}$ is sufficient to mask the OKVS $\mathbf{S}$, and thus it is unnecessary to compute n random vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$. Thus, P_0 needs to generate n VOLE correlations with the same vector $\boldsymbol{u}$ by running a protocol with n different parties. We model this as a multi-party VOLE functionality $\mathcal{F}_{\mathsf{mvole}}$ shown in Figure 3. Note that when P_0 is corrupted, the adversary allows to choose different vectors $\boldsymbol{u}$. That is, we do not require the consistency check of vector $\boldsymbol{u}$. Furthermore, we do *not* require P_0 to broadcast the vector $\boldsymbol{d} = \mathbf{S} - \boldsymbol{u}$. These are sufficient to design multi-party PSI protocols in the malicious setting (see the security proof of our protocol given in Section 4.3). Such a functionality $\mathcal{F}_{\mathsf{mvole}}$ can be UC-realized by calling the standard VOLE functionality (shown in Figure 2) n times and programming the input/output of P_0 to keep the consistency of $\boldsymbol{u}$ in the honest case (see [BCG+19b] for more details).

Furthermore, we further reduce the computation cost of the multi-party PSI protocol described as above by $2n\ell$ H operations. In particular, we construct the PSI protocol by directly using $\mathcal{F}_{\mathsf{mvole}}$ as the underlying primitive instead of the OPRF primitive. We adopt $\mathcal{F}_{\mathsf{mvole}}$ and an OKVS to obtain an OPRF, but simplify the computation of $\mathsf{H}(\mathsf{PRF}_i(h), h) = \mathsf{H}\big(\mathsf{H}(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h), h\big)$ performed by P_i for $i \in [1, n], h \in X_i$ in the previous construction as

$$\mathsf{H}(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h).$$

As such, the computation of $\sum_{i=1}^{n} \mathsf{H}(\mathsf{PRF}_i(h), h) = \sum_{i=1}^{n} \mathsf{H}\big(\mathsf{H}(\mathsf{Decode}(\mathbf{W}_i, h), h), h\big)$ performed by P_0 for $h \in X_0$ can be simplified as

$$\sum_{i=1}^{n} \mathsf{H}(\mathsf{Decode}(\mathbf{W}_i, h), h).$$

In other words, we combine two hash operations into one, which removes the redundant hash operations and significantly improves the computation efficiency.

3.3 Our Approach to Reduce Communication Bandwidth

The state-of-the-art multi-party PSI protocol [GPR+21] adopts the *star* network architecture. In particular, the central party P_0 will have to send the messages $\{\boldsymbol{d}_i = \mathbf{S} - \boldsymbol{u}_i \in \mathbb{F}^m\}_{i \in [1,n]}$ to n parties for computing OPRF values and receive the OKVSs $\mathbf{Q}_1, \ldots, \mathbf{Q}_n \in \mathbb{F}^m$ from n parties. These make the communication bandwidth of P_0 be $nm|\mathbb{F}|$ bits per protocol execution, where $m = \ell/(0.81 - o(1))$ for the recent OKVS scheme [GPR+21] and ℓ is the size of a set. For a large number n of parties and a large size ℓ of input sets, this forms an efficiency bottleneck of the PSI protocol (especially

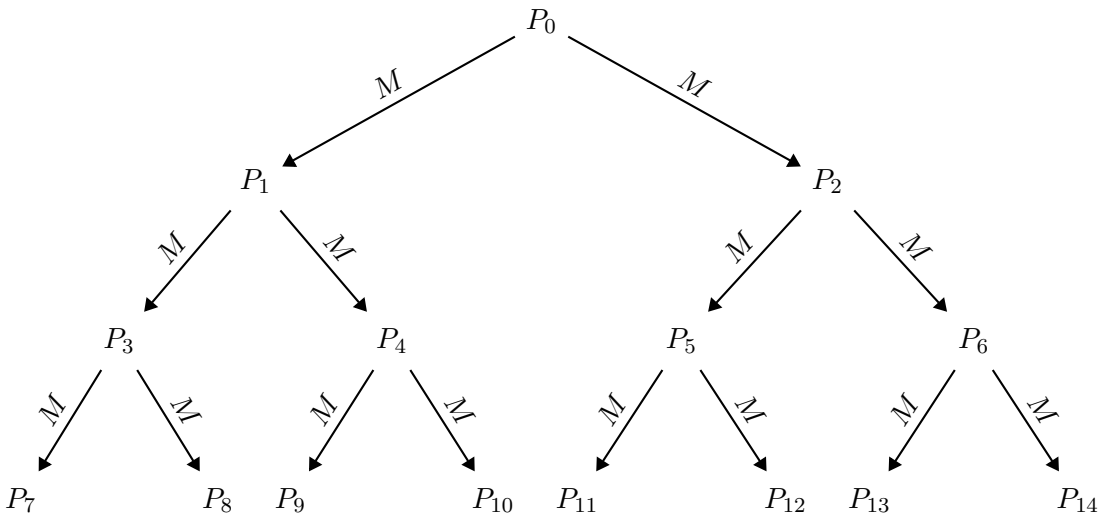

Figure 4: **Transmission of a message M using a binary tree when $n = 14$.**

for networks with restricted bandwidthes). For example, when $\ell = 2^{24}$, $n = 20$ and $|\mathbb{F}| = 128$, P_0 needs about 53 gigabits of communication bandwidth. To reduce the communication bandwidth, one may consider using the Cuckoo hashing approach [KMP+17] to divide a large set into multiple bins. However, this approach is *not* secure against malicious adversaries [NTY21].

Our solution. In the following, we present an efficient approach to reduce the communication bandwidth of P_0 to $2m|\mathbb{F}|$ bits by amortizing the communication among all parties in a *binary-tree* architecture. Specifically, we first apply the optimized approach described in the previous section to the multi-party PSI protocol. In this case, P_0 needs to send the same message $\boldsymbol{d} = \mathbf{S} - \boldsymbol{u}$ to all other parties. Therefore, we can use a binary-tree structure to send $\boldsymbol{d}$. The parties $P_0, P_1, \ldots, P_n$ constitute a binary tree with the root P_0. Party P_0 can only send $\boldsymbol{d}$ to its two child nodes P_1 and P_2. Then, for each level of the tree, every parent node P_{j-1} sends $\boldsymbol{d}$ to its children P_{2j-1} and P_{2j}. See Figure 4 for an example of $n = 14$ where the message $M = \boldsymbol{d}$. Note that here we do *not* require to broadcast vector $\boldsymbol{d}$.

To reduce the bandwidth that communicates the OKVSs $\mathbf{Q}_1, \ldots, \mathbf{Q}_n$, we require that the OKVS scheme is *linear*, which is satisfied by the recent highly-efficient constructions [RS21, GPR+21, RR22]. In this case, we can aggregate these OKVSs into one OKVS through a binary tree. For each level of the tree (from bottom to top), two child nodes P_{2j-1} and P_{2j} respectively send $\mathbf{Q}_{2j-1}$ and $\mathbf{Q}_{2j}$ to their parent node P_{j-1} who updates its OKVS as $\mathbf{Q}_{j-1} := \mathbf{Q}_{j-1} + \mathbf{Q}_{2j-1} + \mathbf{Q}_{2j}$. Finally, the root node P_0 obtains the OKVS $\mathbf{Q}_0 := \sum_{i=0}^{n} \mathbf{Q}_i$. See Figure 5 for an example of $n = 14$, where P_i holds an OKVS $M_i = \mathbf{Q}_i \in \mathbb{F}^m$ for $i \in [0, n]$. According to the linearity of the OKVS scheme, we have $\sum_{i=0}^{n} \mathsf{Decode}(\mathbf{Q}_i, h) = \mathsf{Decode}(\sum_{i=0}^{n} \mathbf{Q}_i, h)$. Therefore, the OKVS $\sum_{i=0}^{n} \mathbf{Q}_i$ that has been aggregated allows P_0 to obtain the correct output. The message-aggregation approach can also be used for the recent multi-party PSI protocols based on garbled Bloom filters [ENOPC21, KMS21], where the approach can reduce the communication bandwidth of the central party P_0 in the protocol [ENOPC21] by a factor of $O(n)$ and the rounds in the protocol [KMS21] from $O(n)$ to $O(\log n)$. In addition, our tree-architecture approach is able to be applied in the multi-party PSI protocol [NTY21] for any corruption threshold $t \leq n$.

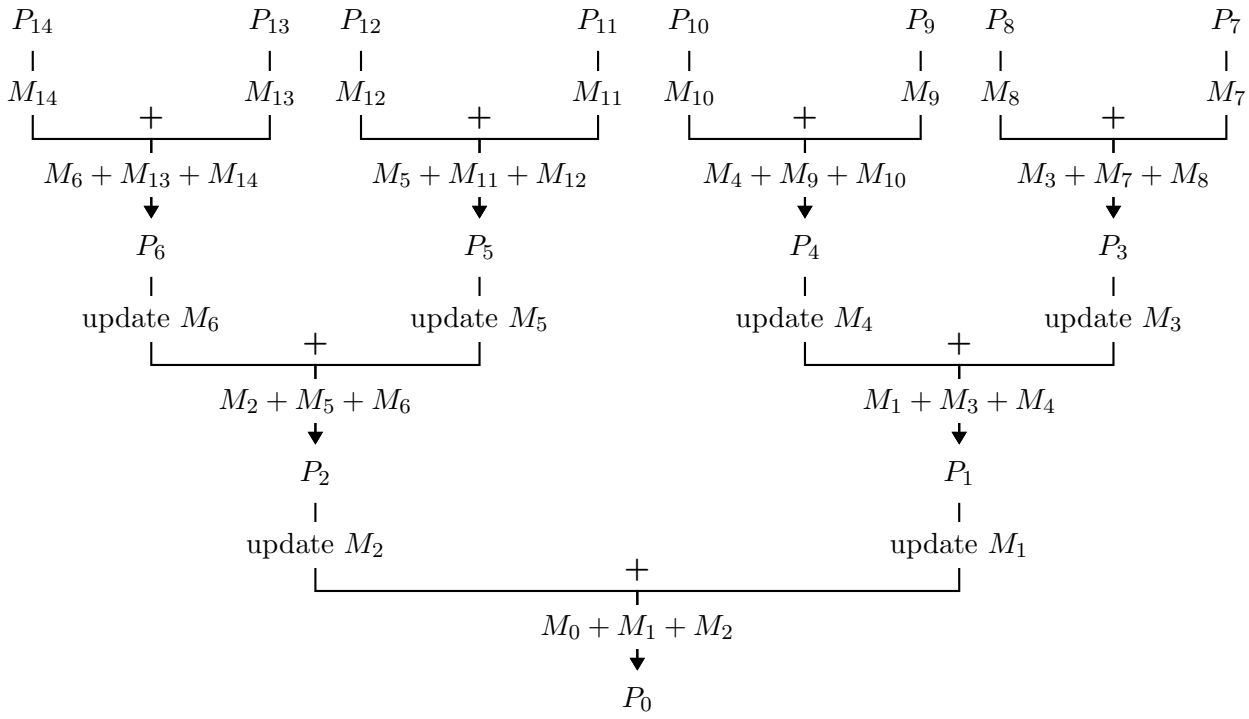

Figure 5: **Message addition aggregation using a binary tree when** $n = 14$

Protocol $\Pi^{\mathsf{tree}}_{\mathsf{send}}$

Parameter: Let $P_0, P_1, ..., P_n$ be $n+1$ parties that execute the protocol. Let $h = \lfloor \log(n+1) \rfloor$.

Input: P_0 holds a message $M \in \{0,1\}^*$. For $i \in [1,n]$, P_i will receive the message M.

Protocol execution:

1. Let M_i be the message received by P_i for $i \in [0,n]$, where $M_0 = M$. For an honest party P_0, we always have that $M_i = M$ for $i \in [1,n]$.
2. For each $i \in [1,h]$, for each $j \in [2^{i-1}, 2^i)$, P_{j-1} sends the message M_{j-1} (received by itself) to P_{2j-1} if $2j-1 \leq n$ and P_{2j} if $2j \leq n$, where P_{j-1} is the parent of two child nodes P_{2j-1} and P_{2j}.

Figure 6: **Protocol for sending messages with constant bandwidth using a binary tree.**

4 Maliciously Secure Multi-Party PSI Protocol

We first describe two sub-protocols which are used to send and aggregate messages respectively in a binary-tree architecture. Then, we present the detailed construction of our multi-party PSI protocol with malicious security. Finally, we give a formal proof of security for our PSI protocol.

4.1 Protocols for Sending and Aggregating Messages Using Tree Architecture

We first describe the sub-protocol $\Pi^{\mathsf{tree}}_{\mathsf{send}}$ shown in Figure 6, which allows the central party P_0 to send a message M to all other parties $P_1, \ldots, P_n$. The message is transmitted in a binary-tree architecture, which enables us to obtain $O(1)$ communication bandwidth instead of $O(n)$. As a trade-off, the round complexity is increased from $O(1)$ to $O(\log n)$. A malicious party P_i for

Protocol $\Pi^{\mathsf{tree}}_{\mathsf{aggregate}}$

Parameter: Let $P_0, P_1, ..., P_n$ be $n+1$ parties that run the protocol. Let $h = \lfloor \log(n+1) \rfloor$.

Inputs: For $i \in [1, n]$, P_i holds a message $M_i \in \mathbb{F}^m$. Party P_0 will obtain the output $\sum_{i\in[1,n]} M_i$.

Protocol execution:

1. P_0 sets $M_0 := 0$.
2. From $i = h$ to $i = 1$, for each $j \in [2^{i-1}, 2^i)$, P_{2j-1} sends M_{2j-1} to P_{j-1} (if $2j-1 \leq n$) and P_{2j} sends M_{2j} to P_{j-1} (if $2j \leq n$), and then P_{j-1} computes $M'_{j-1} := M_{j-1} + M_{2j-1} + M_{2j}$ and updates $M_{j-1} := M'_{j-1}$.
3. P_0 outputs the final message M_0.

Figure 7: **Message aggregate protocol with constant bandwidth using a binary tree.**

$i \in [1, n]$ may send $M' \neq M$ to its left child and $M'' \neq M$ to its right child. In the next subsection, we will show that such malicious behavior is harmless for the security of our PSI protocol.

In Figure 7, we describe the sub-protocol $\Pi^{\mathsf{tree}}_{\mathsf{aggregate}}$, which allows n parties to send the sum of their messages to the central party P_0. These messages are aggregated by addition operations in the binary-tree network architecture, which reduces the communication bandwidth from $O(n)$ to $O(1)$. Similarly, the round complexity is increased from $O(1)$ to $O(\log n)$.

In both of two sub-protocols, the message is always transmitted between a parent node and two child nodes. While the parent node sends the same message to two child nodes in the protocol $\Pi^{\mathsf{tree}}_{\mathsf{send}}$, two child nodes send two different messages to the parent node who sums the two messages and its message as an aggregation message in the protocol $\Pi^{\mathsf{tree}}_{\mathsf{aggregate}}$.

4.2 Our PSI protocol with Efficient Bandwidth and Computation

In Figure 8, we describe the details of our multi-party PSI protocol in the dishonest-majority malicious setting. The communication bandwidth of the central party P_0 is $O(\ell|\mathbb{F}|)$ bits, where the state-of-the-art protocol [GPR+21] that builds on the technique [KMP+17] requires the bandwidth of $O(n\ell|\mathbb{F}|)$ bits for P_0. This protocol works in the $\mathcal{F}_{\mathsf{mvole}}$-hybrid model, and is executed by $n+1$ parties $P_0, P_1, \ldots, P_n$. We separate the protocol into two phases: preprocessing phase where the input sets are unknown, and online phase in which the sets are known. In this protocol, only the central party P_0 obtains the output.

Correctness. To see that the PSI protocol as described in Figure 8 is correct in the honest case, we first note that for any h,

$$\sum_{i=0}^{n} s_h^i = \sum_{i=0}^{n} \left(\sum_{j<i} \mathsf{PRF}(k_{i,j}, h) - \sum_{j>i} \mathsf{PRF}(k_{j,i}, h) \right) = 0.$$

Then, we observe that for each $h \in \bigcap_{i\in[0,n]} X_i$, for each $i \in [1, n]$, the following holds:

$$\begin{aligned}
z_{i,h} - z'_{i,h} &= \mathsf{H}(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h) - \mathsf{H}(\mathsf{Decode}(\mathbf{W}_i, h), h) \\
&= \mathsf{H}(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h) - \mathsf{H}(\mathsf{Decode}(\mathbf{V}_i + \mathbf{S} \cdot \Delta_i, h), h) \\
&= \mathsf{H}(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h) - \mathsf{H}(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{Decode}(\mathbf{S}, h) \cdot \Delta_i, h) \\
&= \mathsf{H}(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h) - \mathsf{H}(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h) = 0.
\end{aligned}$$

Protocol Π_{mpsi}

Parameters:

- A large finite field $\mathbb{F}$ with $|\mathbb{F}| \geq 2^\kappa$.
- Linear OKVS scheme $(\mathsf{Encode}, \mathsf{Decode})$ that maps ℓ items to m slots.
- A cryptographic hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{F}$ modeled as a random oracle.
- Let $\mathsf{PRF} : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \mathbb{F}$ be a Pseudo-Random Function (PRF).

Inputs: For $n+1$ parties $P_0, P_1, ..., P_n$, every party P_i holds a set X_i such that $X_i \in \mathbb{F}$ and $|X_i| = \ell$. Let P_0 be the designated party to receive the output $\bigcap_{i\in[0,n]} X_i$.

Preprocessing: This procedure can be executed when the sets are *unknown*.

1. Every party P_i samples $k_{i,j} \leftarrow \{0,1\}^\kappa$ as a random PRF key, and then sends it to P_j over a private channel for each $j < i$. (The zero-sharing setup phase is run only once.)
2. For each $i \in [1,n]$, P_i sends (init, i) to the multi-party VOLE functionality $\mathcal{F}_{\mathsf{mvole}}$ (shown in Figure 3), which returns Δ_i to P_i.
3. The parties $P_0, P_1, ..., P_n$ send (extend, m) to functionality $\mathcal{F}_{\mathsf{mvole}}$, which returns $\boldsymbol{u} \in \mathbb{F}^m$ and $\{\boldsymbol{w}_i\}_{i\in[1,n]}$ to P_0, and outputs $\boldsymbol{v}_i \in \mathbb{F}^m$ and $\Delta_i \in \mathbb{F}$ to P_i for each $i \in [1,n]$, such that $\boldsymbol{w}_i = \boldsymbol{v}_i + \boldsymbol{u} \cdot \Delta_i$ for $i \in [1,n]$.

Online: This procedure is run when the sets are *known*.

4. P_0 computes OKVS via
$$\mathbf{S} = \mathsf{Encode}\left(\{(h, \mathsf{H}(h)) \,|\, h \in X_0\}\right),$$
and calculates $\boldsymbol{d} = \mathbf{S} - \boldsymbol{u} \in \mathbb{F}^m$. Then P_0 sends $\boldsymbol{d}$ to P_i for each $i \in [1,n]$ by running the sub-protocol $\Pi^{\mathsf{tree}}_{\mathsf{send}}$ (shown in Figure 6) in a bandwidth-efficient way.
5. For each $i \in [1,n]$, P_i computes $\mathbf{V}_i := \boldsymbol{v}_i - \boldsymbol{d} \cdot \Delta_i$. P_0 sets $\mathbf{W}_i := \boldsymbol{w}_i$ for $i \in [1,n]$ where $\mathbf{W}_i = \mathbf{V}_i + \mathbf{S} \cdot \Delta_i$.
6. For each $i \in [0,n]$, for each $h \in X_i$, every party P_i computes the following:
$$s^i_h := \sum_{j<i} \mathsf{PRF}(k_{i,j}, h) - \sum_{j>i} \mathsf{PRF}(k_{j,i}, h).$$
7. For each $i \in [1,n]$, P_i computes $z_{i,h} = \mathsf{H}(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h)$ for each $h \in X_i$, and then computes OKVS $\mathbf{Q}_i$ via
$$\mathbf{Q}_i = \mathsf{Encode}\left(\left\{(h, z_{i,h} + s^i_h) \,|\, h \in X_i\right\}\right).$$
8. Parties $P_1, ..., P_n$ run the sub-protocol $\Pi^{\mathsf{tree}}_{\mathsf{aggregate}}$ (shown in Figure 7) to send $\mathbf{Q} = \sum_{i=1}^n \mathbf{Q}_i \in \mathbb{F}^m$ to P_0 in a bandwidth-efficient way.
9. P_0 computes $z'_{i,h} = \mathsf{H}(\mathsf{Decode}(\mathbf{W}_i, h), h)$ for each $h \in X_0$, and then outputs
$$\left\{ h \in X_0 \;\middle|\; \mathsf{Decode}(\mathbf{Q}, h) + s^0_h - \sum_{i=1}^n z'_{i,h} = 0 \right\}.$$

Figure 8: **Maliciously secure multi-party PSI protocol in the $\mathcal{F}_{\mathsf{mvole}}$-hybrid model.**

Therefore, for each $h \in \bigcap_{i\in[0,n]} X_i$, we have

$$\begin{aligned}
\mathsf{Decode}(\mathbf{Q}, h) + s^0_h - \sum_{i=1}^n z'_{i,h} &= \mathsf{Decode}(\sum_{i=1}^n \mathbf{Q}_i, h) + s^0_h - \sum_{i=1}^n z'_{i,h} \\
&= s^0_h + \sum_{i=1}^n \mathsf{Decode}(\mathbf{Q}_i, h) - \sum_{i=1}^n z'_{i,h} \\
&= s^0_h + \sum_{i=1}^n (z_{i,h} + s^i_h) - \sum_{i=1}^n z'_{i,h}
\end{aligned}$$

4.3 Proof of Security

In the following, we prove the security of our PSI protocol in the multi-party malicious setting. Our proof of security will use the following lemma, which has been proven in [GPR+21].

Lemma 1 ([GPR+21]). *Given a set of parties that run the zero-sharing setup (i.e., the step 1 in Figure 8) such that a pair of parties P_i, P_j are honest and the adversary's view is independent of the P_i's share s_h^i, then the P_j's share s_h^j is computationally indistinguishable from a uniform value.*

Based on the above lemma, we prove the following theorem.

Theorem 1. *Let* H *be a random oracle and* $(\mathsf{Encode}, \mathsf{Decode})$ *be a linear OKVS scheme. Let* PRF *be a pseudo-random function. Then protocol* Π_{mpsi} *shown in Figure 8 UC-realizes functionality* $\mathcal{F}_{\mathsf{mpsi}}$ *in the presence of a malicious adversary corrupting up to n of the $n+1$ parties in the* $\mathcal{F}_{\mathsf{mvole}}$*-hybrid model.*

Proof. We first consider the case of a malicious central party P_0, and then consider the case of an honest P_0. In each case, we construct a PPT simulator $\mathcal{S}$ given access to functionality $\mathcal{F}_{\mathsf{mpsi}}$, running a PPT adversary $\mathcal{A}$ as a subroutine when emulating $\mathcal{F}_{\mathsf{mvole}}$. For each case, $\mathcal{S}$ simulates the random oracle H by responding the queries made by $\mathcal{A}$ with random values when keeping the consistency of responses. We always implicitly consider that whenever $\mathcal{A}$ aborts, $\mathcal{S}$ sends abort to functionality $\mathcal{F}_{\mathsf{mpsi}}$ and then aborts. For each case, we show that no PPT environment $\mathcal{Z}$ can distinguish the real-world execution from the ideal-world execution. Let $\mathcal{C} \subset [0, n]$ denote the set of corrupted parties.

Malicious central party P_0. When emulating $\mathcal{F}_{\mathsf{mvole}}$, $\mathcal{S}$ interacts with adversary $\mathcal{A}$ as follows:

1. In the zero-sharing setup, on behalf of every honest party P_i, $\mathcal{S}$ samples $k_{i,j} \leftarrow \{0,1\}^\kappa$ and sends it to $\mathcal{A}$ for each $j < i, j \in \mathcal{C}$, as well as receives $k_{j,i} \in \{0,1\}^\kappa$ from $\mathcal{A}$ for each $j \in \mathcal{C}, j > i$. Thus, for any h, $\mathcal{S}$ can compute the share s_h^i for $i \notin \mathcal{C}$, when it also samples the PRF keys between any two honest parties.

2. $\mathcal{S}$ emulates the (init) command of $\mathcal{F}_{\mathsf{mvole}}$ and receives Δ_i for $i \in \mathcal{C}$ from $\mathcal{A}$. Then, $\mathcal{S}$ emulates the (extend) command of $\mathcal{F}_{\mathsf{mvole}}$ by receiving $\boldsymbol{u}_i, \boldsymbol{w}_i \in \mathbb{F}^m$ for $i \in [1, n]$ from $\mathcal{A}$. For $i \in [1, n]$, $\mathcal{S}$ sets $\mathbf{W}_i = \boldsymbol{w}_i$.

3. During the protocol execution of $\Pi_{\mathsf{send}}^{\mathsf{tree}}$, $\mathcal{S}$ receives $\boldsymbol{d}_i \in \mathbb{F}^m$ for $i \notin \mathcal{C}$ from $\mathcal{A}$, and then computes $\mathbf{S}_i := \boldsymbol{d}_i + \boldsymbol{u}_i$ for $i \notin \mathcal{C}$. Then, for $i \notin \mathcal{C}$, $\mathcal{S}$ computes the following set

$$X_0^i := \{h \mid \mathcal{A} \text{ queried } \mathsf{H} \text{ at } h \text{ and } \mathsf{Decode}(\mathbf{S}_i, h) = \mathsf{H}(h)\}.$$

4. $\mathcal{S}$ sends $\tilde{X}_0 := \bigcap_{i \notin \mathcal{C}} X_0^i$ to functionality $\mathcal{F}_{\mathsf{mpsi}}$, as the input of all corrupted parties, and then receives $Y = \tilde{X}_0 \cap (\bigcap_{i \notin \mathcal{C}} X_i)$ as the output.

5. For each $i \notin \mathcal{C}$, $\mathcal{S}$ chooses an arbitrary different $h \notin Y$ from the domain of X_i for each unknown value in set $X_i \backslash Y$, and then computes the OKVS $\mathbf{Q}_i$ of honest party P_i as follows:

$$\mathbf{Q}_i = \begin{cases} \mathsf{Encode}\big(h, \mathsf{H}(\mathsf{Decode}(\mathbf{W}_i, h), h) + s_h^i\big) & \text{if } h \in Y \\ \mathsf{Encode}(h, r_{i,h}) \text{ with } r_{i,h} \leftarrow \mathbb{F} & \text{if } h \notin Y \end{cases} \tag{1}$$

   On behalf of every honest party P_i, $\mathcal{S}$ sends $\mathbf{Q}_i$ to $\mathcal{A}$ when simulating the protocol execution of $\Pi_{\mathsf{aggregate}}^{\mathsf{tree}}$.

Hybrid argument. In the following, we use a series of hybrid games to prove that the real-world execution is computationally indistinguishable from the ideal-world execution.

$\mathbf{Hybrid}_0$: This is the real-world execution, where every honest party P_i computes the OKVS $\mathbf{Q}_i$ as follows:

$$\mathbf{Q}_i = \mathsf{Encode}\Big(\Big\{\big(h, \mathsf{H}(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h)\cdot\Delta_i, h) + s_h^i\big) \;\Big|\; h \in X_i\Big\}\Big).$$

$\mathbf{Hybrid}_1$: In this hybrid, the simulator $\mathcal{S}$ simulates the zero-sharing setup honestly, and emulates functionality $\mathcal{F}_{\mathsf{mvole}}$ by recording the values sent by $\mathcal{A}$ as described above.

Note that for each $i \in \mathcal{C} \cap [1, n]$, letting $\mathcal{A}$ choose the output vector $\boldsymbol{w}_i$ for corrupted party P_0 is equivalent to making $\mathcal{A}$ choose $\boldsymbol{v}_i$ for corrupted party P_i, as $\boldsymbol{w}_i = \boldsymbol{v}_i + \boldsymbol{u}_i \cdot \Delta_i$ holds. Thus, $\mathbf{Hybrid}_1$ has the identical distribution as $\mathbf{Hybrid}_0$.

$\mathbf{Hybrid}_2$: In this hybrid, $\mathcal{S}$ extracts the sets X_0^i for all $i \notin \mathcal{C}$ as described above. Then, for each $i \notin \mathcal{C}$, $\mathcal{S}$ computes $T_i := X_i \cap X_0^i$, where the simulator knows the input sets of honest parties in the hybrid. On behalf of every honest party P_i, $\mathcal{S}$ computes the OKVS $\mathbf{Q}_i$ as follows:

$$\mathbf{Q}_i = \begin{cases} \mathsf{Encode}\big(h, \mathsf{H}(\mathsf{Decode}(\mathbf{W}_i, h), h) + s_h^i\big) & \text{if } h \in T_i \\ \mathsf{Encode}(h, r_{i,h}) \text{ with } r_{i,h} \leftarrow \mathbb{F} & \text{if } h \in X_i \backslash T_i \end{cases}$$

Then, for $i \notin \mathcal{C}$, $\mathcal{S}$ sends $\mathbf{Q}_i$ to $\mathcal{A}$ in the tree-aggregation way during the execution of $\Pi^{\mathsf{tree}}_{\mathsf{aggregate}}$.

From the simulation, we can see that $\mathbf{W}_i = \mathbf{V}_i + \mathbf{S}_i \cdot \Delta_i$ for $i \notin \mathcal{C}$. From the linearity of the Decode algorithm, for $h \in X_0^i$, we have the following:

$$\begin{aligned} \mathsf{Decode}(\mathbf{W}_i, h) &= \mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{Decode}(\mathbf{S}_i, h) \cdot \Delta_i \\ &= \mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i. \end{aligned}$$

Therefore, for $i \notin \mathcal{C}$, for $h \in T_i = X_i \cap X_0^i$, the corresponding element of $\mathbf{Q}_i$ in $\mathbf{Hybrid}_2$ has the identical distribution as that in $\mathbf{Hybrid}_1$. Below, we analyze the case of $h \in X_i \backslash T_i$. While $\mathbf{Q}_i = \mathsf{Encode}\big(h, \mathsf{H}(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h)\cdot\Delta_i, h) + s_h^i\big)$ in $\mathbf{Hybrid}_1$, $\mathbf{Q}_i = \mathsf{Encode}(h, r_{i,h})$ in $\mathbf{Hybrid}_2$. From the above equation, we know that $\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i = \mathsf{Decode}(\mathbf{W}_i, h) + (\mathsf{H}(h) - \mathsf{Decode}(\mathbf{S}_i, h)) \cdot \Delta_i$ for $h \notin X_0^i$. Since $\mathsf{Decode}(\mathbf{S}_i, h) \neq \mathsf{H}(h)$ and $\Delta_i \in \mathbb{F}$ is uniformly random from the adversary's view, the probability that $\mathcal{A}$ makes a query $(\mathsf{Decode}(\mathbf{W}_i, h) + (\mathsf{H}(h) - \mathsf{Decode}(\mathbf{S}_i, h)) \cdot \Delta_i, h)$ to random oracle H is at most $q/|\mathbb{F}|$ that is negligible in security parameter κ, where q is the number of H queries. Therefore, for $h \notin X_0^i$, $\mathsf{H}(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h) = \mathsf{H}(\mathsf{Decode}(\mathbf{W}_i, h) + (\mathsf{H}(h) - \mathsf{Decode}(\mathbf{S}_i, h)) \cdot \Delta_i, h)$ is uniform from the adversary's view in the random oracle model. Further, we have that the distribution of the elements of $\mathbf{Q}_i$ associated with $h \in X_i \backslash T_i$ in $\mathbf{Hybrid}_1$ is computationally indistinguishable from that in $\mathbf{Hybrid}_2$.

$\mathbf{Hybrid}_3$: In this hybrid, $\mathcal{S}$ changes the way that creates the OKVS $\mathbf{Q}_i$ of every honest party P_i as follows:

$$\mathbf{Q}_i = \begin{cases} \mathsf{Encode}\big(h, \mathsf{H}(\mathsf{Decode}(\mathbf{W}_i, h), h) + s_h^i\big) & \text{if } h \in \bigcap_{i \notin \mathcal{C}} T_i \\ \mathsf{Encode}(h, r_{i,h}) \text{ with } r_{i,h} \leftarrow \mathbb{F} & \text{if } h \in X_i \backslash \bigcap_{i \notin \mathcal{C}} T_i \end{cases} \tag{2}$$

Clearly, if there is only one honest party, then the distribution of $\mathbf{Hybrid}_3$ is identical to that of $\mathbf{Hybrid}_2$. In the following, we assume that there exists at least two honest parties. It is easy to see that if $h \in \bigcap_{i \notin \mathcal{C}} T_i$, then $\mathbf{Hybrid}_3$ is identical to $\mathbf{Hybrid}_2$. For each $i \notin \mathcal{C}$, the computation of $\mathbf{Q}_i$ is different in two hybrids for a value $h \in T_i$, where $h \notin T_j$ for some different

$j \notin \mathcal{C}$. When $h \notin T_j$, we note that in $\mathbf{Hybrid}_2$, the computation of $\mathbf{Q}_j$ does not use the value s_h^j anywhere. From Lemma 1, we have that s_h^i is computationally indistinguishable from a uniformly random value. Thus, for $h \in T_i$ and $h \notin T_j$, the view of $\mathsf{Encode}\big(h, \mathsf{H}(\mathsf{Decode}(\mathbf{W}_i, h), h) + s_h^i\big)$ is indistinguishable from $\mathsf{Encode}(h, r_{i,h})$ for a uniform value $r_{i,h} \in \mathbb{F}$. Therefore, the distribution of $\mathbf{Hybrid}_3$ is computationally indistinguishable from that of $\mathbf{Hybrid}_2$.

$\mathbf{Hybrid}_4$: In this hybrid, $\mathcal{S}$ computes $\tilde{X}_0 := \bigcap_{i \notin \mathcal{C}} X_0^i$, and then receives the output $Y = \tilde{X}_0 \cap (\bigcap_{i \notin \mathcal{C}} X_i)$ from functionality $\mathcal{F}_{\mathsf{mpsi}}$. For each $i \notin \mathcal{C}$, $\mathcal{S}$ chooses an arbitrary different $h \notin Y$ as each unknown value in set $X_i \backslash Y$, and then computes the OKVS $\mathbf{Q}_i$ of honest party P_i as follows:

$$\mathbf{Q}_i = \begin{cases} \mathsf{Encode}\big(h, \mathsf{H}(\mathsf{Decode}(\mathbf{W}_i, h), h) + s_h^i\big) & \text{if } h \in Y \\ \mathsf{Encode}(h, r_{i,h}) \text{ with } r_{i,h} \leftarrow \mathbb{F} & \text{if } h \notin Y \end{cases}$$

On behalf of every honest party P_i, $\mathcal{S}$ sends $\mathbf{Q}_i$ to $\mathcal{A}$ during the protocol execution of $\Pi^{\mathsf{tree}}_{\mathsf{aggregate}}$.

It is easy to see that $Y = \tilde{X}_0 \cap (\bigcap_{i \notin \mathcal{C}} X_i) = \bigcap_{i \notin \mathcal{C}} (\tilde{X}_0 \cap X_i) = \bigcap_{i \notin \mathcal{C}} T_i$. Thus, for each $h \in Y$, the distribution of $\mathbf{Q}_i$ in $\mathbf{Hybrid}_3$ is identical to that in $\mathbf{Hybrid}_4$. For the case of $h \notin Y$, the only difference between $\mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_4$ is that $h \in X_i \backslash Y$ in $\mathbf{Hybrid}_3$, while h is chosen arbitrarily from the domain in $\mathbf{Hybrid}_4$. From the security of the OKVS scheme $(\mathsf{Encode}, \mathsf{Decode})$, we have that the only difference is negligible in κ.

Honest central party P_0. Given access to functionality $\mathcal{F}_{\mathsf{mpsi}}$, $\mathcal{S}$ interacts with $\mathcal{A}$ as follows:

1. $\mathcal{S}$ simulates the zero-sharing setup just as in the case of malicious P_0, and thus can compute the share s_h^i for any h and each $i \notin \mathcal{C}$.

2. Similarly, $\mathcal{S}$ emulates the (init) command of $\mathcal{F}_{\mathsf{mvole}}$ and receives Δ_i for $i \in \mathcal{C}$ from $\mathcal{A}$. Then, $\mathcal{S}$ emulates the (extend) command of $\mathcal{F}_{\mathsf{mvole}}$ by receiving $\boldsymbol{v}_i \in \mathbb{F}^m$ for $i \in \mathcal{C}$ from $\mathcal{A}$.

3. On behalf of honest party P_0, $\mathcal{S}$ samples a uniform vector $\boldsymbol{d} \leftarrow \mathbb{F}^m$, and then sends it to $\mathcal{A}$ during the protocol execution of $\Pi^{\mathsf{tree}}_{\mathsf{send}}$.

4. For $i \in \mathcal{C}$, $\mathcal{S}$ computes $\mathbf{V}_i = \boldsymbol{v}_i - \boldsymbol{d} \cdot \Delta_i \in \mathbb{F}^m$ following the protocol description.

5. If there are at least two honest parties, for each $i \notin \mathcal{C}, i \neq 0$, $\mathcal{S}$ chooses an arbitrary input set X_i', and samples $r_{i,h} \leftarrow \mathbb{F}$ for $h \in X_i'$. Then, for $i \notin \mathcal{C}, i \neq 0$, $\mathcal{S}$ computes the OKVS $\mathbf{Q}_i$ as

   $$\mathbf{Q}_i = \mathsf{Encode}\left(\left\{(h, r_{i,h}) \,|\, h \in X_i'\right\}\right).$$

   On behalf of every honest party P_i $(i \neq 0)$, $\mathcal{S}$ sends $\mathbf{Q}_i$ during the protocol execution of $\Pi^{\mathsf{tree}}_{\mathsf{aggregate}}$. Note that in the binary-tree architecture, the OKVS $\mathbf{Q}_i$ from an honest party P_i may be sent to adversary $\mathcal{A}$ who corrupts the parent node P_j.

6. From the protocol execution of $\Pi^{\mathsf{tree}}_{\mathsf{aggregate}}$, $\mathcal{S}$ computes the sum of OKVSs $\sum_{i \in \mathcal{C}} \mathbf{Q}_i$.

7. According to the queries of random oracle H, $\mathcal{S}$ defines the set

   $$O = \Big\{h \,\Big|\, \mathcal{A} \text{ made a query of the form } \mathsf{H}\big(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h\big) \text{ for } i \in \mathcal{C}\Big\},$$

   and then computes

   $$\tilde{X} := \Big\{h \in O \,\Big|\, \mathsf{Decode}(\sum_{i \in \mathcal{C}} \mathbf{Q}_i, h) = \sum_{i \in \mathcal{C}} \mathsf{H}\big(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h\big) - \sum_{i \notin \mathcal{C}} s_h^i\Big\}.$$

8. $\mathcal{S}$ sends $\tilde{X}$ to functionality $\mathcal{F}_{\mathsf{mpsi}}$ as the input for every corrupted party. Then, $\mathcal{F}_{\mathsf{mpsi}}$ returns the output $\tilde{X} \cap (\bigcap_{i \notin \mathcal{C}} X_i)$ to honest party P_0.

Hybrid argument. In the following, we use a series of hybrid games to prove that the real-world execution is computationally indistinguishable from the ideal-world execution.

Hybrid$_0$: This is the real-world execution, where honest party P_0 computes the output as:

$$\Big\{h \in X_0 \,\Big|\, \sum_{i \in \mathcal{C}} \mathsf{Decode}(\mathbf{Q}_i, h) + \sum_{i \notin \mathcal{C}} \mathsf{Decode}(\mathbf{Q}_i, h) + s_h^0 - \sum_{i=1}^{n} \mathsf{H}\big(\mathsf{Decode}(\mathbf{W}_i, h), h\big) = 0\Big\}.$$

Hybrid$_1$: In this hybrid, how P_0 computes the output is modified as:

$$\Big\{h \in \bigcap_{i \notin \mathcal{C}} X_i \,\Big|\, \sum_{i \in \mathcal{C}} \mathsf{Decode}(\mathbf{Q}_i, h) + \sum_{i \notin \mathcal{C}} \mathsf{Decode}(\mathbf{Q}_i, h) + s_h^0 - \sum_{i=1}^{n} \mathsf{H}\big(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h\big) = 0\Big\}.$$

From the linearity of Decode, we have that for $i \in [1, n]$, for $h \in X_0 \subseteq \bigcap_{i \notin \mathcal{C}} X_i$,

$$\begin{aligned}
\mathsf{Decode}(\mathbf{W}_i, h) &= \mathsf{Decode}(\mathbf{V}_i + \mathbf{S} \cdot \Delta_i, h) \\
&= \mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{Decode}(\mathbf{S}, h) \cdot \Delta_i \\
&= \mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i.
\end{aligned}$$

Therefore, the computation of the output only differs for values $h \in X_0$ where $h \notin X_i$ for some $i \notin \mathcal{C}$. Such an h will never satisfy the equation in **Hybrid$_1$** as $h \notin X_i$. For such an h, since $h \notin X_i$, $\mathbf{Q}_i$ is generated independently of s_h^i. Nevertheless, the output computed by P_0 at h contains a term s_h^0, which renders the whole equation uniformly random in the computational sense (by Lemma 1). Thus, the probability that h satisfies the equality in **Hybrid$_0$** is at most $1/|\mathbb{F}|$ which is negligible in κ.

Note that if $h \in \bigcap_{i \notin \mathcal{C}} X_i$, then $\mathsf{Decode}(\mathbf{Q}_i, h) = \mathsf{H}\big(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h\big) + s_h^i$ for $i \notin \mathcal{C}$. Therefore, we can rewrite the equation as follows:

$$\Big\{h \in \bigcap_{i \notin \mathcal{C}} X_i \,\Big|\, \sum_{i \in \mathcal{C}} \mathsf{Decode}(\mathbf{Q}_i, h) = \sum_{i \in \mathcal{C}} \mathsf{H}\big(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h\big) - \sum_{i \notin \mathcal{C}} s_h^i\Big\}.$$

Hybrid$_2$: In this hybrid, how P_0 computes the output is modified again. Specifically, the simulator computes the following set

$$O = \Big\{h \,\Big|\, \mathcal{A} \text{ made a query of the form } \mathsf{H}\big(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h\big) \text{ for } i \in \mathcal{C}\Big\}.$$

Then, on behalf of P_0, the simulator computes the output with a modified computation as follows:

$$\Big\{h \in O \cap (\bigcap_{i \notin \mathcal{C}} X_i) \,\Big|\, \sum_{i \in \mathcal{C}} \mathsf{Decode}(\mathbf{Q}_i, h) = \sum_{i \in \mathcal{C}} \mathsf{H}\big(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h\big) - \sum_{i \notin \mathcal{C}} s_h^i\Big\}.$$

It suffices to show that if $h \notin O$, then the probability that h would have satisfied the equation in **Hybrid$_1$** is negligible in κ. Indeed, if $h \notin O$, then when the adversary sends $\mathbf{Q}_i$ for $i \in \mathcal{C}$, the value $\mathsf{H}\big(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h\big)$ is distributed independently of the adversary's view. Since the condition in **Hybrid$_1$** is a linear equality including the term, the probability that it is satisfied is at most $1/|\mathbb{F}|$ which is negligible in κ. We can rewrite the above equation as

$$\Big\{h \in O \,\Big|\, \sum_{i \in \mathcal{C}} \mathsf{Decode}(\mathbf{Q}_i, h) = \sum_{i \in \mathcal{C}} \mathsf{H}\big(\mathsf{Decode}(\mathbf{V}_i, h) + \mathsf{H}(h) \cdot \Delta_i, h\big) - \sum_{i \notin \mathcal{C}} s_h^i\Big\} \cap (\bigcap_{i \notin \mathcal{C}} X_i) = \tilde{X} \cap (\bigcap_{i \notin \mathcal{C}} X_i).$$

Hybrid$_3$: In this hybrid, the simulator $\mathcal{S}$ simulates the zero-sharing setup honestly, and emulates functionality $\mathcal{F}_{\mathsf{mvole}}$ as described above, as well as sends a uniform vector $\boldsymbol{d} \in \mathbb{F}^m$ to adversary $\mathcal{A}$ via executing protocol $\Pi^{\mathsf{tree}}_{\mathsf{send}}$ with $\mathcal{A}$.

Clearly, the simulation of the zero-sharing setup and $\mathcal{F}_{\mathsf{mvole}}$ is perfect. In **Hybrid$_2$**, the vector $\boldsymbol{d} = \mathbf{S} - \boldsymbol{u}$ is uniformly random, as $\boldsymbol{u}$ is uniform in the presence of the adversary's view. Therefore, **Hybrid$_3$** has the identical distribution as **Hybrid$_2$**.

Hybrid$_4$: In this hybrid, if there exists at least two honest parties, then for each $i \notin \mathcal{C}, i \neq 0$, $\mathcal{S}$ computes and sends the OKVS $\mathbf{Q}_i$ to $\mathcal{A}$ via running protocol $\Pi^{\mathsf{tree}}_{\mathsf{aggregate}}$ with $\mathcal{A}$, in the way that an arbitrary input set X'_i and random values $r_{i,h} \in \mathbb{F}$ for $h \in X'_i$ are used in the computation of OKVS $\mathbf{Q}_i$ as described above.

If there are at least two honest parties, the simulator needs to simulate the OKVS of honest parties before seeing the OKVS of corrupted parties, as the OKVS is transmitted in a binary-tree architecture. For each $i \notin \mathcal{C}, i \neq 0$, for each $h \in X_i$, the value $z_{i,h} + s^i_h$ is uniformly random in **Hybrid$_3$**, since s^i_h is uniform for the case of honest P_i and P_0 from the adversary's view. From the security of the OKVS scheme $(\mathsf{Encode}, \mathsf{Decode})$, $\mathcal{A}$ cannot distinguish between the real input set X_i and the dummy input set X'_i after seeing the OKVS $\mathbf{Q}_i$ for each $i \notin \mathcal{C}, i \neq 0$. Overall, **Hybrid$_4$** is computationally indistinguishable from **Hybrid$_3$**.

Hybrid$_5$: In this hybrid, $\mathcal{S}$ extracts the set $\tilde{X}$ following the way as described above, and then sends $\tilde{X}$ to functionality $\mathcal{F}_{\mathsf{mpsi}}$ as the input for every corrupted party.

According to the definition of the output of P_0 in the previous hybrid, it is easy to see that **Hybrid$_5$** has the identical distribution as **Hybrid$_4$**, where P_0 will obtain the same output in two hybrids.

In conclusion, no PPT environment $\mathcal{Z}$ can distinguish between the real-world execution and the ideal-world execution, which completes the proof. □

5 An Attack against Multi-Output Extension of A PSI Protocol

In the malicious setting, it is a non-trivial task to extend a multi-party PSI protocol to support multiple outputs where every party (instead of only party P_0) will obtain the output. This is because the parties cannot be trusted to deliver the intersection output faithfully. In the multi-party malicious setting, Garimella et al. [GPR+21] extended their PSI protocol with a single output to a protocol supporting multiple outputs that achieves the best efficiency. In this section, we present a simple and practical attack for the multi-output extension [GPR+21], which allows the attacker to reveal more information of the sets of honest parties than that obtained from the intersection of the sets of all parties. Specifically, we first review the multi-output extension by Garimella et al. [GPR+21]. This extension modifies the multi-party PSI protocol shown in Section 3.1 to realize that all parties obtain the output in the following procedure:

- All parties $P_0, P_1, \ldots, P_n$ publicly commit to their OKVSs $\{\mathbf{Q}_i\}_{i\in[0,n]}$ respectively. That is, every party P_i broadcasts a commitment $com_i = \mathsf{Commit}(\mathbf{Q}_i; r_i)$ to all other parties where r_i is a randomness.
- After all the commitments that have been made, the parties open these commitments. That is, for $i \in [0, n]$, every party P_i sends $(\mathbf{Q}_i, r_i)$ to all other parties, and then verifies the correctness of all the openings (i.e., checking that $com_j = \mathsf{Commit}(\mathbf{Q}_j; r_j)$ for all $j \neq i$).

- Every party P_i with $i \in [0, n]$ computes the output as

$$\Big\{h \in X_i \,\Big|\, \sum_{j=0}^{n} \mathsf{Decode}(\mathbf{Q}_j, h) = 0\Big\}.$$

An attack to leak the information of the sets held by honest parties. Below, we show a simple but practical attack against the above multi-output extension to reveal more information of the sets held by honest parties beyond the intersection of all sets. Suppose that P_0 is honest. Let $\mathcal{C} \subseteq [1, n]$ denote the set of corrupted parties. In particular, an adversary $\mathcal{A}$ (who even behaves semi-honestly) is able to perform the following attack:

1. $\mathcal{A}$ receives the OKVS $\mathbf{Q}_i$ for all $i \notin \mathcal{C}$ that contain $\mathbf{Q}_0$. Recall that the OKVS $\mathbf{Q}_i$ for $i \notin \mathcal{C}, i \neq 0$ is defined as follows:

$$\mathbf{Q}_i = \mathsf{Encode}\Big(\Big\{\big(h, \mathsf{H}(\mathsf{PRF}_i(h), h) + s_h^i\big) \,\Big|\, h \in X_i\Big\}\Big).$$

   The OKVS $\mathbf{Q}_0$ is computed as

$$\mathbf{Q}_0 = \mathsf{Encode}\Big(\Big\{\big(h, s_h^0 - \sum_{i=1}^{n} \mathsf{H}(\mathsf{PRF}_i(h), h) \,\Big|\, h \in X_0\big)\Big\}\Big).$$

2. $\mathcal{A}$ computes $\mathbf{Q} := \sum_{i \notin \mathcal{C}} \mathbf{Q}_i$. According to the definition of zero sharings, we know that $\sum_{i \in [0,n]} s_h^i = 0$ for any h. Therefore, for each $h \in \bigcap_{i \notin \mathcal{C}} X_i$, we obtain the following:

$$\sum_{i \notin \mathcal{C}} \mathsf{Decode}(\mathbf{Q}_i, h) = -\sum_{i \in \mathcal{C}} s_h^i - \sum_{i \in \mathcal{C}} \mathsf{H}(\mathsf{PRF}_i(h), h). \tag{3}$$

3. $\mathcal{A}$ who corrupts P_i for $i \in \mathcal{C}$ can compute the shares s_h^i and values $\mathsf{PRF}_i(h)$ for all $i \in \mathcal{C}$ when any h is known. Then, for any $h \notin \bigcap_{i \in [0,n]} X_i$, $\mathcal{A}$ is able to decide whether $h \in \bigcap_{i \notin \mathcal{C}} X_i$ by checking if the equation (3) holds. For the items in the set $\bigcap_{i \notin \mathcal{C}} X_i$ that have a low entropy, $\mathcal{A}$ can directly reveal these items by enumerating the items and then checking their correctness, and thus obtain the secret data included in $\bigcap_{i \notin \mathcal{C}} X_i$.

In the special case that only P_0 is honest and all other parties are corrupted (i.e., $|\mathcal{C}| = n$), the adversary can leak some secret items in the set X_0 by performing the above attack. The reason behind the successful attack is that the OKVS $\mathbf{Q}_0$ is revealed by the honest central party P_0. This does not occur in the original multi-party PSI protocol [GPR+21] that only P_0 obtains the output, where P_0 only uses the OKVS $\mathbf{Q}_0$ *locally* in the original protocol. To prevent the attack as describe above, one may need to change the way computing the OKVS $\mathbf{Q}_0$ and introduce some high-entropy secrets into $\mathbf{Q}_0$. We leave it as an interesting future work to design a concretely efficient multi-party PSI protocol supporting multiple outputs in the malicious setting.

Acknowledgements

Work of Kang Yang is supported by the National Natural Science Foundation of China (Grant Nos. 62102037, 61932019). Work of Yu Yu is supported by the National Key Research and Development Program of China (Grant Nos. 2020YFA0309705 and 2018YFA0704701) and the National Natural Science Foundation of China (Grant Nos. 62125204 and 61872236). Yu Yu also acknowledges the support from the XPLORER PRIZE. We thank anonymous reviewers for their helpful comments.

References

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM Conf. on Computer and Communications Security (CCS) 2019*, pages 291–308. ACM Press, 2019.

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *Advances in Cryptology—Crypto 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, 2019.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. pages 1069–1080. IEEE, 2020.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *ACM Conf. on Computer and Communications Security (CCS) 2018*, pages 896–912. ACM Press, 2018.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE, 2001.

[CDG+21] Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. Efficient linear multiparty PSI and extensions to circuit/quorum PSI. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security – CCS'21*, pages 1182–1204. Association for Computing Machinery, 2021.

[CJS10] Jung Hee Cheon, Stanislaw Jarecki, and Jae Hong Seo. Multi-party privacy-preserving set intersection with quasi-linear complexity. Cryptology ePrint Archive, Report 2010/512, 2010. https://eprint.iacr.org/2010/512.

[CM20] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In *Advances in Cryptology—Crypto 2020, Part III*, LNCS, pages 34–63. Springer, 2020.

[CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. LNCS, pages 502–534. Springer, 2021.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *ACM Conf. on Computer and Communications Security (CCS) 2013*, pages 789–800. ACM Press, 2013.

[DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. Pir-psi: Scaling private contact discovery. *Proc. Priv. Enhancing Technol.*, 2018(4):159–178, 2018.

[ENOPC21] Aner Ben Efraim, Olga Nissenbaum, Eran Omri, and Anat Paskin-Cherniavsky. Psimple: Practical multiparty maliciously-secure private set intersection. Cryptology ePrint Archive, Report 2021/122, 2021. https://ia.cr/2021/122.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology—Eurocrypt 2004*, LNCS, pages 1–19. Springer, 2004.

[GN19] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. In *Advances in Cryptology—Eurocrypt 2019, Part III*, volume 11478 of *LNCS*, pages 154–185. Springer, 2019.

[GPR+21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. LNCS, pages 395–425. Springer, 2021.

[HV17] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In *Intl. Conference on Theory and Practice of Public Key Cryptography 2017, Part I*, LNCS, pages 175–203. Springer, 2017.

[IKN+17] Mihaela Ion, Ben Kreuter, Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. Private intersection-sum protocol with applications to attributing aggregate ad conversions. Cryptology ePrint Archive, Report 2017/738, 2017. https://eprint.iacr.org/2017/738.

[IKN+20] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 370–389, 2020.

[IOP18] Roi Inbar, Eran Omri, and Benny Pinkas. Efficient scalable multiparty private set-intersection via garbled bloom filters. In *Intl. Conf. on Security and Cryptography for Networks (SCN)*, LNCS, pages 235–252. Springer, 2018.

[KMP+17] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *ACM Conf. on Computer and Communications Security (CCS) 2017*, pages 1257–1272. ACM Press, 2017.

[KMS21] Alireza Kavousi, Javad Mohajeri, and Mahmoud Salmasizadeh. Efficient scalable multi-party private set intersection using oblivious PRF. Cryptology ePrint Archive, Report 2021/484, 2021. https://ia.cr/2021/484.

[KRS+19] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 1447–1464, 2019.

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *Advances in Cryptology—Crypto 2005*, volume 3621 of *LNCS*, pages 241–257. Springer, 2005.

[NT21] Duong Tung Nguyen and Ni Trieu. Mpccache: Privacy-preserving multi-party cooperative cache sharing at the edge. Cryptology ePrint Archive, Report 2021/317, 2021. https://ia.cr/2021/317.

[NTY21] Ofri Nevo, Ni Trieu, and Avishay Yanai. Simple, fast malicious multiparty private set intersection. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer*

and Communications Security, CCS'21, page 1151–1165. Association for Computing Machinery, 2021.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In *Advances in Cryptology—Crypto 2019, Part III*, volume 11694 of *LNCS*, pages 401–431. Springer, 2019.

[PRTY20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In *Advances in Cryptology—Eurocrypt 2020, Part II*, volume 12106 of *LNCS*, pages 739–767. Springer, 2020.

[RR17] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In *Advances in Cryptology—Eurocrypt 2017, Part I*, volume 10210 of *LNCS*, pages 235–259. Springer, 2017.

[RR22] Peter Rindal and Srinivasan Raghuraman. Blazing fast psi from improved okvs and subfield vole. Cryptology ePrint Archive, Report 2022/320, 2022. https://ia.cr/2022/320.

[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. LNCS, pages 901–930. Springer, 2021.

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In *ACM Conf. on Computer and Communications Security (CCS) 2019*, pages 1055–1072. ACM Press, 2019.

[SS07] Yingpeng Sang and Hong Shen. Privacy preserving set intersection protocol secure against malicious behaviors. In *Proceedings of the Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies – PDCAT'07*, page 461–468. IEEE Computer Society, 2007.

[SS08] Yingpeng Sang and Hong Shen. Privacy preserving set intersection based on bilinear groups. In *the 31th Australasian Computer Science Conference – ACSC 2008*, volume 74, pages 47–54. Australian Computer Society, 2008.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1074–1091, 2021.

[YWL+20] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *ACM Conf. on Computer and Communications Security (CCS) 2020*, pages 1607–1626. ACM Press, 2020.

[ZLL+19] En Zhang, Feng-Hao Liu, Qiqi Lai, Ganggang Jin, and Yu Li. Efficient multi-party private set intersection against malicious adversaries. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop – CCSW'19*, pages 93–104. Association for Computing Machinery, 2019.

A OKVS Overfitting

For the security proof of a maliciously secure PSI protocol, the simulator obtains an OKVS from a corrupted party, and needs to extract the keys that are encoded in the OKVS. In general, this is done by defining $v_i = \mathsf{H}(k_i)$ for $i \in [1, \ell]$ where H is a random oracle. Then, the simulator can observe the queries to H made by the adversary, and then check which of the keys k satisfy $\mathsf{Decode}(\mathbf{S}, k) = \mathsf{H}(k)$. An OKVS whose parameters are chosen to encode ℓ keys may often hold even more than ℓ keys, when it is generated by the adversary. In the context of PSI, this allows the adversary to encode more keys than advertised. Therefore, we need to bound the number of keys that the adversary can "overfit" into an OKVS. Following the previous work [GPR+21], we model the property in the following definition.

Definition 4 ([GPR+21])**.** *The (ℓ, ℓ')-OKVS overfitting game is defined as follows.*

- *Let* $(\mathsf{Encode}, \mathsf{Decode})$ *be an OKVS with parameters chosen to support ℓ items, and $\mathcal{A}$ be any PPT adversary. Let $\mathsf{H} : \mathcal{K} \to \mathcal{V}$ be a random oracle.*
- *Run* $\mathbf{S} \leftarrow \mathcal{A}^{\mathsf{H}(\cdot)}(1^\kappa)$.
- *Define the following set:*

$$X = \{k \,|\, \mathcal{A} \text{ made a query } (k) \text{ to } \mathsf{H} \text{ and } \mathsf{Decode}(\mathbf{S}, k) = \mathsf{H}(k)\}.$$

- *If $|X| > \ell'$, then the adversary $\mathcal{A}$ wins.*

We say that the (ℓ, ℓ')-OKVS overfitting problem is hard for an OKVS, if no PPT adversary wins this game except with negligible probability.

For $\kappa = 128$ and $\lambda = 40$, according to the analysis [PRTY20], when $\mathsf{H} : \mathcal{K} \to \mathbb{F}$ is used to define the values, a linear OKVS with a field size $|\mathbb{F}| = 128$ can guarantee that the successful probability of the adversary in the above overfitting game is less than $1/2^{40}$, even though the adversary is allowed to make 2^{80} queries to H.