Maliciously Secure Circuit-PSI via SPDZ-Compatible Oblivious PRF

Yaxi Yang*, Xiaojian Liang†, Xiangfu Song‡, Linting Huang§, Hongyu Ren§, Changyu Dong§, Jianying Zhou*
*Singapore University of Technology and Design,†Independent Researcher,
‡National University of Singapore,§Guangzhou University,
Email: {yaxi_yang,jianying_zhou}@sutd.edu.sg, im.liangxj@gmail.com
songxf@comp.nus.edu.sg, linting_huang@e.gzhu.edu.cn
lubricantrhy@gmail.com,changyu.dong@gmail.com

***Abstract*—Circuit Private Set Intersection (Circuit-PSI) allows two parties to compute any functionality f on items in the intersection of their input sets without revealing any information about the intersection set. It is a well-known variant of PSI and has numerous practical applications. However, existing circuit-PSI protocols only provide security against *semi-honest* adversaries. One straightforward solution is to extend a pure garbled-circuit-based PSI (NDSS'12) to a maliciously secure circuit-PSI, but it will result in non-concrete complexity. Another is converting state-of-the-art semi-honest circuit-PSI protocols (EUROCRYPT'21; PoPETS'22) to be secure in the malicious setting. However, it will come across *the consistency issue* since parties can not guarantee the inputs of functionality f stay unchanged as obtained from the last step.**

This paper addresses the aforementioned issue by introducing the first maliciously secure circuit-PSI protocol. The central building block named Distributed Dual-key Oblivious PRF (DDOPRF), provides an oblivious evaluation of secret-shared inputs with dual keys. Additionally, we ensure the compatibility of DDOPRF with SPDZ, enhancing the versatility of our circuit-PSI protocol. Notably, our construction allows us to guarantee fairness within circuit-PSI effortlessly. Importantly, our circuit-PSI protocol also achieves online linear computation and communication complexities.

I. Introduction

The private set intersection (PSI) aims to compute the intersection set of two sets of items from two parties without revealing any other information [19, 23, 27]. PSI has plenty of applications, including contact tracing [20], advertising conversion [54], secure data analysis [3], and genomic sequence testing [53]. Many companies utilize PSI for commercial activities or database transmissions [24, 35]. Consequently, they contribute to the development of numerous open-source projects related to PSI, such as SecretFLow[1] from Antgroup, Private-Join-and-Compute[2] from Google. In addition to PSI, parties typically would like to generalize the standard PSI protocol to meet practical needs. An interesting application is f(PSI), which usually called circuit-PSI [11, 23]. Circuit-PSI aims to compute some arbitrary functionality f over the intersection set, and only reveal the computation result f(PSI), but not the items in the intersection themselves [30]. However, existing works for circuit-PSI [11, 23, 41, 44, 47] are under semi-honest model, whose security properties may not hold in the presence of malicious adversaries. Designing a circuit-PSI protocol in the malicious model is very meaningful as it captures many realistic scenarios where the parties may take arbitrary strategies to break the security of a protocol.

Huang et al. [23] leveraged a pure garbled-circuit method to design a sort-and-compare network to compute the intersection set. A possible way to extend [23] with malicious security is via a general malicious garbled-circuit method, e.g., authenticated garbling method [51]. However, the computational complexity depends on the intricacy of the functionality that parties need to compute and is linear in the number of non-linear gates in the circuit. This can result in impractical computational cost and circuit complexity.

For semi-honest circuit-PSI protocols [11, 41, 44, 47], they designed Oblivious PRF (OPRF) protocols based on Oblivious Transfer (OT) to compute the shares of the intersection set instead of letting parties learn the intersection set in the clear. The parties can use the secret-shared set as input for the following secure computation. However, if those circuit-PSI protocols are transformed into malicious versions by substituting all underlying protocols into malicious ones, it will cause *a consistency issue*. That is to say, when two parties obtain secret-shared values and send them to the following computation, the malicious adversary may change the values. Previous work has not considered how to ensure that parties use the same values they obtained from the last step for the input of functionality of f(PSI).

A straightforward way to solve this issue is to use a commitment or authentication protocol to ensure that the values sent to the functionality f stay unchanged. Miao et al. [34] utilize a Pedersen commitment, additive homomorphic encryption, and zero-knowledge proof protocol to achieve a PSI-sum protocol (a specialization of f(PSI)) in the malicious setting. However, their protocol is subject to expensive asymmetric operations and causes higher complexity, and it only supports computing the sum of intersection payload rather than arbitrary computations. Therefore, this leaves the following open problem:

Can we solve the consistency issue and construct a malicious circuit-PSI protocol with linear computation and communication complexity?

We answer this question affirmatively by proposing the first

[1]https://github.com/secretflow
[2]https://github.com/google/private-join-and-compute

malicious circuit-PSI protocol with concrete efficiency. The first solution that comes to our mind is to use a generalized secure computation method, SPDZ [15, 26], as an underlying primitive. SPDZ supports secret-shared secure addition and multiplication over a finite field and uses the Message Authentication Code (MAC) to authenticate each input value. It can be seen as a general run-time MPC environment. However, if we use SPDZ in a black box way, and parties call the underlying primitives in SPDZ to achieve the same functionality as f(PSI), it may need plenty of addition and multiplication operations and result in higher communication rounds. Instead of using a general framework to solve this problem, we redesign different modules in circuit-PSI to integrate a succinct malicious circuit-PSI protocol. The contributions of this paper are as follows.

Our Contributions. We design a Distributed Dual-key Oblivious Pseudorandom Function (DDOPRF) protocol. It aims to obliviously compute the PRF results for secret-shared input sets $\vec{x}$ and $\vec{y}$, and distributed keys. Then, parties can compare the PRF results and select the secret-shared input sets to the following functionality f computation to get $f(\vec{x} \cap \vec{y})$. During this process, we need to ensure that parties can not learn which items are selected and used as input of f. Therefore, parties will use a malicious secret-shared shuffle protocol to shuffle their input items.

Initially, we intended to design a succinct distributed OPRF protocol to achieve f(PSI) and make it more practical, and it fulfilled our intention as expected. Then, we found that our designed OPRF protocol can be extended with a "dual-key" gadget, which only has constant communication extra costs. And this gadget can bring fairness to the protocol. In more detail, we summarise important features of our protocol as follows:

- **Malicious Security.** DDOPRF is compiled with SPDZ, so based on the MAC authentication method of SPDZ, our circuit-PSI protocol provides malicious security properties. Compared to the previous efficient circuit-PSI protocols [11, 30, 44, 47], we are the first to propose a circuit-PSI protocol in the malicious setting without any third party.
- **Fairness.** To achieve fairness in a malicious protocol can be costly and hard to achieve based on cryptographic protocols [18]. It needs to prevent the corrupted party from aborting the protocol prematurely after obtaining the output (and before the other party obtains it). Fairness can be crucial in many motivating scenarios where both parties need to know the intersection set, particularly in computational scenarios between companies. The right of one company that does not receive the computation results will be violated if the PSI protocol is single-output. Some fair PSI protocols have been proposed [1, 16, 18]. However, those PSI protocols are subjected to low efficiency or need a trusted third party. Therefore, we design a "dual-key" mechanism and use it as a module in our circuit-PSI protocol to achieve fairness. The augmentation of the circuit-PSI protocol with fairness incurs only minimal computational overhead.
- **Linear Complexity.** We build our circuit-PSI protocol based on the DDOPRF protocol, which is succinct and only needs two-round communication to get PRF results for the input values. The "dual-key" mechanism is simple and costs constant communication and computation overheads. We formalize each module according to our malicious circuit-PSI protocol, achieving linear complexity.

In conclusion, we propose the first maliciously secure circuit-PSI protocol based on an SPDZ-compatible OPRF protocol (DDOPRF). The code of our paper is valuable at https://github.com/mcPSI.

A. Application

The limitation of the previous PSI-related works is that those are specially designed for intersection set computation. Our circuit-PSI protocol can be seen as a secret-shared computation with more versatility and applicability. Some of the applications are listed as follows.

Application 1: Database operations.
By plugging our SPDZ-compatible OPRF into the set computations, we obtain various two-party private set computation protocols that are secure in the malicious setting. Private set computation is crucial for database operations It can be used for operations on the database in a privacy-preserving way, for example, database join [54], query [20]. Specifically, the two companies have two databases respectively, and they want to perform joint operations on the items of the two databases. They can use circuit-PSI to align items according to indexes, select the items in both databases, or perform any operations on the selected items.

Application 2: Vertical Federated Learning (VFL).
VFL [32] is a privacy-preserving machine learning framework in which the training dataset is vertically partitioned. So items in the dataset share the same ID while holding different features. PSI-related protocols can be used for ID alignment in a VFL system. Parties can get all the items with the same ID while keeping the other items private.

Application 3: Privacy-preserving telemetry.
Telemetry refers to the collection and transmission of various types of data from different devices, sensors, or equipment of systems, such as a shipboard system, to a central monitoring station. Those data can be analyzed and help in real-time monitoring, control, and analysis of various parameters related to the ship's operation, safety, and performance [37, 45]. However, if the telemetry data comes from different service manufacturers, the data needs to be treated as private and not be revealed to others during the analysis. Circuit-PSI can be used to identify the identities and counts of all items distributed from different parties [22], and it is also a most powerful way to visualize data and is thus used in many data analysis applications [33].

II. Related Work

Before diving into the technical details of our construction, we compare our protocol to relevant works in this section and discuss challenges in extending the existing PSI solutions to the malicious circuit-PSI problem.

Malicious PSI. The most common method to achieve malicious PSI protocols [9, 19, 40, 44, 46, 47] is utilizing the Oblivious Key-Value Store (OKVS) structures. Dong et al. [19] propose a semi-honest two-party PSI protocol based on Garbled Bloom Filters (GBF), one of OKVS structures. Based on Dong et al.'s [19] work, Rindal et al. [46] convert it to a malicious setting via a cut-and-choose technique. Next, Pinkas et al. [40] propose the first two-party PSI protocol with linear communication and security in the malicious setting. In their work, an OKVS structure based on cuckoo hashing is proposed and achieves a constant rate. Then, Rindal et al. [47] optimize the OKVS structure by combining VOLE, and achieve the performance improvement compared to Pinkas et al. [40]. Furthermore, Bienstock et al. [9] present an RB-OKVS scheme, which achieves the best encoding rate (0.97) and the best efficiency compared to priors OKVS structures. Plugging the RB-OKVS scheme into the PSI implementation [44], it obtains the most efficient malicious PSI to date. However, if we trivially extend malicious PSI to circuit-PSI, that is to say, the parties send the PSI results to the following functionality, it will reveal $\{\vec{x} \cap \vec{y}\}$ to parties.

Circuit-PSI. The functionality of circuit-PSI is to securely compute arbitrary functions over the intersection set. Huang et al. [23] present the notion of circuit-PSI, and use a generic garbled-circuit approach [55] to achieve it. It achieves $O(nlogn)$ complexity with small constant factors, where n is the size of the input set. Afterward, Pinkas et al. [41] propose a circuit-PSI protocol based on OPRF and reduce the communication complexity to $O(n)$. However, the computational complexity of their protocol is super-linear $O(n(logn)^2)$. While this bottleneck is solved in [11], Chandran et al. propose a concretely efficient circuit-PSI protocol with linear complexity. Both protocols [11, 41] are based on the IKNP-style OT extension protocol [4], and the communication cost of those can be improved by utilizing the Vector Oblivious Linear Evaluation (VOLE) style OT extensions as discussed in [44, 47]. However, it will involve more computation cost, and the concrete performance depends on the network parameters [11]. Specifically, those circuit-PSI protocols [11, 41, 44, 47] are secure in the semi-honest setting.

The core idea of the OPRF-based circuit-PSI protocols [11, 41] is similar to PSI protocols [27, 42] except that the intersection results are secret-shared between the parties, which can be used as inputs of the following circuit computation. In more detail, for a value v_0 (resp. v_1) in the input set of P_0 (resp. P_1), P_0 (resp. P_1) will gets an output random value a_0 (resp. a_1). If $v_0 = v_1$, and v_0 is in the intersection set, then we can get $1 = a_0 \oplus a_1$. Otherwise, $0 = a_0 \oplus a_1$. As we can see, if we adopt those circuit-PSI protocols to the malicious setting, the main challenge is how to guarantee the consistency of secret-shared results and inputs of the following circuits. Since the intersection set results are secret-shared between two parties, a malicious party might tamper with the secret-shared results and send the tampered results to the following circuits. Then, the correctness of those circuit-PSI results cannot be guaranteed. Next, the protocols in [30] also achieve malicious circuit-PSI with the help of an untrusted third party, and it also reveals the size of the intersection set to the untrusted party.

Fair PSI. Debnath et al. and Dong et al. [16, 18] propose fair two-party PSI protocols, where both parties learn the result or neither does. This is called "all or nothing". Their protocol relies on homomorphic encryption, zero-knowledge proofs, and other asymmetric key primitives. Therefore, those protocols are subjected to the efficiency issue. Abadi et al. [1] extend the fair two-party PSI protocol to a multi-party setting. However, they do not present any experiments or implementation results related to their proposed protocol.

Oblivious Pseudorandom Function. OPRF is an essential primitive for building PSI-related protocols. The frequently used method for building OPRF is based on OT extension protocol [4]. As we mentioned above, a line of circuit-PSI works [13, 27, 28, 39, 41] based on OT extension are subject to the consistency issue when trivially converting those protocols into malicious circuit-PSI. Dodis-Yampolskiy PRF (DY-PRF) [17] is another method to construct OPRF [10]. The DY-PRF-based OPRF can be combined with cryptographic commitment protocols and serve as "glue" for other parts of a circuit-PSI protocol to solve the consistency issue in the malicious setting. Miao et al. [34] combine a DY-PRF-based OPRF protocol with a Pedersen commitment protocol and achieve a PSI-sum protocol in the malicious setting. Their DY-PRF is built by an additively homomorphic encryption scheme. Therefore, their protocol is subject to the efficiency limitation of HE. In our paper, we take advantage of the secret sharing and authentication methods [26] to avoid the costliest part of their [34] protocol.

III. Preliminaries

We use $\mathbb{G}$ to denote an abelian group, and $\mathbb{F}$ denotes a finite field (*e.g.*, $\mathbb{F} = \mathbb{F}_{p^k}$ for some prime p) with items of ℓ bits. $[n]$ denotes the set $\{0, 1, ..., n-1\}$ and $[l, r]$ to denote $\{l, l+1, ..., r-1, r\}$. Given a set $\vec{x}$, we use $x \xleftarrow{\$} \vec{x}$ to denote x is uniformally sampled from $\vec{x}$. We use $a||b$ to denote strings concatenation of a and b. For an ℓ-bits string $x \in \{0,1\}^\ell$, we use b_i to denote its i-th bit of x, and $x = \sum_{i=1}^{\ell} b_i \cdot 2^{\ell-i}$. For a sharing of x over $\mathbb{G}$, the bit decomposition operation is a protocol for converting a sharing $\langle x \rangle$ into ℓ shares $\langle b_1 \rangle, ..., \langle b_\ell \rangle$, where $\langle b_1 \rangle$ represents the most left bit share of x.

A. Security Model and Fairness Definition

We consider a two-party model in Section IV. In the two-party model, any party can be corrupted by a malicious adversary. We prove the security of our protocols in the ideal/real world paradigm [31]. To begin with, we testify that

our protocol is secure in the semi-honest setting. Then, we compile our protocol to be secure in the malicious setting. To prove the security of our circuit-PSI protocol, the standard functionality of each sub-protocol used in our protocol is presented for access as a trusted party, and to function as a sub-functionality.

As for the definition of fairness, *the complete fairness* is impossible for the two-party setting [14], instead, *partial fairness* can be achieved. Therefore, we achieve a relaxation of fairness (i.e. *partial fairness*) in this paper. We follow and extend the definition of fairness in [7, 38] as follows.

Definition 1. *A two-party secure protocol* Π *that achieves the functionality* $F(x, y)$ *is* (c, ϵ)*-fair if: For any working time* t*, an adversary* A *runs the protocol* Π *for computing* F*. Whenever* A *aborts the protocol and attempts to recover* $F(x, y)$*, let* q_0 *denote the* A*'s probability of success. Then, the other party* C *can run in the working time* $c \cdot t$ *for computing* $F(x, y)$ *after the protocol is aborted by* A*, such that* q_1 *is the* C*'s probability of success. It holds that* $|q_0 - q_1| \leq \epsilon$*.*

In this paper, we consider a relaxation of fairness, which means the adversary has *one-bit* privilege as the upper bound to recover the protocol results. Looking ahead, whenever the protocol aborts, the possibility of one party infers the results $F(x, y)$ with a one-bit advantage over the other party during the same working time.

B. Dodis-Yampolskiy PRF

The Dodis-Yampolskiy PRF (DY-PRF) requires a cyclic group $\mathbb{G}$ with prime-order p, and is defined as

$$F_{\mathrm{DY}}(k, x) = g^{\frac{1}{k+x}},$$

where g is a generator of $\mathbb{G}$, and $k \xleftarrow{\$} \mathbb{Z}_p$. The pseudorandomness is guaranteed by the q-DDH inversion (q-DDHI) problem.

C. Authenticated Secret Sharing

Linear secret sharing. We use $[\![x]\!]$ to denote an additive linear secret sharing (LSS) for $x \in \mathbb{F}$ shared between n parties such that each P_i holds a random share $[\![x]\!]_i \in \mathbb{F}$ with $\sum_{i\in[n]} [\![x]\!]_i = x$. The secret x can be constructed iff all the parties reveal their shares and then sum them up. Therefore, LSS preserves perfect privacy against $n-1$ corrupted parties [6]. If x and y are two values shared between n parties, LSS supports the following linear operations:

- $[\![z]\!] \leftarrow [\![x]\!] + [\![y]\!]$: P_i computes $[\![z]\!]_i \leftarrow [\![x]\!]_i + [\![y]\!]_i$;
- $[\![z]\!] \leftarrow c + [\![x]\!]$: P_0 computes $[\![z]\!]_0 \leftarrow c + [\![x]\!]_0$ and P_i computes $[\![z]\!]_i \leftarrow [\![x]\!]_i$ for all $i \in [n] \setminus \{0\}$;
- $[\![z]\!] \leftarrow c \cdot [\![x]\!]$: P_i computes $[\![z]\!]_i \leftarrow c \cdot [\![x]\!]_i$,

where we can verify $[\![x + y]\!] = [\![x]\!] + [\![y]\!]$, $[\![c + x]\!] = c + [\![x]\!]$, and $[\![c \cdot x]\!] = c \cdot [\![x]\!]$. All the operations mentioned above do not need interaction between parties. In particular, if we want to compute multiplication operation as $[\![z]\!] \leftarrow [\![x]\!] \cdot [\![y]\!]$ and verify $z = xy$, then the parties require interaction. One commonly used approach to achieve multiplication operation is Beaver's method [43]. In detail, suppose the parties pre-share a *Beaver Triple* $([\![a]\!], [\![b]\!], [\![c]\!])$ with $a \cdot b = c$. The parties can perform the following interaction to compute $[\![x \cdot y]\!]$ from $[\![x]\!]$ and $[\![y]\!]$:

- The parties compute $[\![e]\!] \leftarrow [\![x]\!] - [\![a]\!]$ and $[\![f]\!] \leftarrow [\![y]\!] - [\![b]\!]$;
- The parties open $[\![e]\!]$ and $[\![f]\!]$ to obtain e and f;
- The parties compute $[\![z]\!] \leftarrow [\![c]\!] + f \cdot [\![a]\!] + e \cdot [\![b]\!] + e \cdot f$,

where we can verify that $z = xy$ is just as required.

Authenticated secret sharing. Authenticated secret sharing (ASS) ensures the integrity of shared secrets. A typical SPDZ-style ASS [15] relies on *information-theoretic message authentication codes* (IT-MACs) for integrity. To be specific, the parties will additionally share $[\![\xi]\!]$ for a secret MAC key $\xi \xleftarrow{\$} \mathbb{F}$. For a sharing $[\![x]\!]$, the parties also share its MAC sharing $[\![\gamma(x)]\!]$ such that $\gamma(x) = \xi \cdot x$. We call $\langle x \rangle = ([\![x]\!], [\![\gamma(x)]\!])$ as an *authenticated secret sharing* for a secret x, and $\langle x \rangle_i = ([\![x]\!]_i, [\![\gamma(x)]\!]_i) \in \mathbb{F}^2$ as an *authenticated share* held by P_i. Since the soundness error is proportional to the inverse of the field size, we require $\mathbb{F}$ to be sufficiently large (*i.e.*, $|\mathbb{F}| > 2^\kappa$); this is crucial to detect errors with overwhelming probability. ASS supports the following local computation:

- $\langle z \rangle \leftarrow \langle x \rangle + \langle y \rangle$: $\langle z \rangle \leftarrow ([\![x]\!] + [\![y]\!], [\![\gamma(x)]\!] + [\![\gamma(y)]\!])$;
- $\langle z \rangle \leftarrow c + \langle x \rangle$: $\langle z \rangle \leftarrow (c + [\![x]\!], c \cdot [\![\xi]\!] + [\![\gamma(x)]\!])$;
- $\langle z \rangle \leftarrow c \cdot \langle x \rangle$: $\langle z \rangle \leftarrow (c \cdot [\![x]\!], c \cdot [\![\gamma(x)]\!])$,

where we can verify $\langle x + y \rangle = \langle x \rangle + \langle y \rangle$, $\langle c + x \rangle = c + \langle x \rangle$, and $\langle c \cdot x \rangle = c \cdot \langle x \rangle$.

Commonly, the parties may reveal an ASS $\langle x \rangle$ when using ASS sharing for computation, and the parties have to make sure that x is opened correctly. To secure open an ASS sharing $\langle x \rangle$, the parties can leverage the embedded MAC to detect any introduced error. Specifically, the parties compute

$$[\![d]\!] \leftarrow [\![\gamma(x)]\!] - x \cdot [\![\xi]\!]. \tag{1}$$

The parties then each commit to its share of d followed by opening to check if $d = 0$ and abort it is not the case.

Computing multiplication between ASS sharings $\langle x \rangle$ and $\langle y \rangle$ can be done by using an *Authenticated Beaver Triple* $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ satisfying $a \cdot b = c$. The parties can perform the following interaction to compute $\langle x \cdot y \rangle$ from $\langle x \rangle$ and $\langle y \rangle$:

- The parties compute $\langle e \rangle \leftarrow \langle x \rangle - \langle a \rangle$ and $\langle f \rangle \leftarrow \langle y \rangle - \langle b \rangle$;
- The parties *partically* open $\langle e \rangle$ and $\langle f \rangle$ (not their MACs) to obtain e and f;
- The parties compute $\langle z \rangle \leftarrow \langle c \rangle + f \cdot \langle a \rangle + e \cdot \langle b \rangle + e \cdot f$.

In the malicious setting, the corrupted parties may tamper their values when opening e and f. Thus, the parties must check the correct opening of e and f, using the method as shown previously in Eq. (1). Note that the above definitions for LSS and ASS generally work over vectors. We use $[\![\vec{x}]\!]$ to denote a vector sharing of $\vec{x}$, and $\gamma(\vec{x})$ to denote its MAC vector sharing where $\gamma(\vec{x}_i) = \xi \cdot \vec{x}_i$.

Arithmetic black-box. We define the functionality of arithmetic black-box to capture the above commands over ASS sharings over $\mathbb{Z}_p$ as shown in Fig. 1. We refer to well-known instantiations from existing SPDZ-style protocols [15, 25, 26, 36]. For completeness, we also provide the details in §A.

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

Parameters: a prime p.

The ABB functionality contains the following commands:

- $\langle r \rangle \leftarrow \mathsf{Rand}()$: Output an ASS sharing $\langle r \rangle$ for $r \in \mathbb{Z}_p$.
- $\langle x \rangle \leftarrow \mathsf{Input}(x)$: Output a randomly ASS sharing $\langle x \rangle$ for the input value x.
- $(\langle a \rangle, \langle b \rangle, \langle c \rangle) \leftarrow \mathsf{RandomMul}()$: Output three ASS sharings $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ such that $a \cdot b = c$.
- $\langle z \rangle \leftarrow \mathsf{Mul}(\langle x \rangle, \langle y \rangle)$: On input $\langle x \rangle$ and $\langle y \rangle$, output $\langle z \rangle$ such that $z = x \cdot y$.
- Linear combination: Given $\langle x \rangle, \langle y \rangle$ and $a, b, c \in \mathbb{Z}_p$, the parties can compute $\langle z \rangle = a \cdot \langle x \rangle + b \cdot \langle y \rangle + c$ for free with communication.
- $x \leftarrow \mathsf{Open}(\langle x \rangle)$: On input an ASS sharing $\langle x \rangle$, open x to all the parties, and check the MAC value of x.

Fig. 1. The arithmetic black-box functionality.

D. Permutation

A permutation is a bijective function $\pi : [n] \mapsto [n]$. We use $\mathbf{S}_n$ to denote a symmetric group containing all $[n] \mapsto [n]$ permutations. For a vector $\vec{x} = \{x_1, ..., x_n\}$, when a permutation function π is applied over $\vec{x}$, the value x_i ($i \in [n]$) is moved to the position $\pi(i)$ as

$$\vec{y} = \pi(\vec{x}) = (\vec{x}_{\pi(0)}, \cdots, \vec{x}_{\pi(n-1)}). \tag{2}$$

Then, we use π^{-1} to denote the inverse of a permutation π. Therefore, $\vec{y}_i = \vec{x}_{\pi(i)}$, or equivalently, $\vec{x}_i = \vec{y}_{\pi^{-1}(i)}$. We denote by $\pi \circ \rho$ the composition of two permutations π and ρ such that $\pi \circ \rho(i) = \pi(\rho(i))$.

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

Parameters: a prime p; n denotes the dimension of the shared vector to be shuffled.

The SSS functionality contains the following commands:

- $\mathsf{Shuffle}$: On input $\langle \vec{x} \rangle$ with $\vec{x} \in \mathbb{Z}_p^n$, sample a random permutation $\pi \xleftarrow{\$} \mathbf{S}_n$. Compute $\vec{x'} \leftarrow \pi(\vec{x})$ and reshare $\langle \vec{x'} \rangle$ between the parties.

Fig. 2. The ideal secret-shared shuffle functionality.

E. Secret-shared shuffle

A secret-shared shuffle (SSS) with authentication allows the shareholders to jointly permutate authenticated secret sharing $\langle \vec{x} \rangle$ of a vector $\vec{x}$ using a random permutation π known by neither party [12]. This paper will rely on a maliciously secure SSS ideal functionality over ASS sharings. The functionality is formally defined in Fig. 2. The detail of the SSS protocol [49] used in our circuit-PSI protocol is described in Appendix B.

IV. Construction

In this section, we first provide an overview and the intuition of our proposed malicious circuit-PSI protocol Π_{mcPSI}. Next, we introduce our proposed sub-protocol, DDOPRF, and explain how DDOPRF is used as the main building block of Π_{mcPSI} to fulfill the privacy requirements.

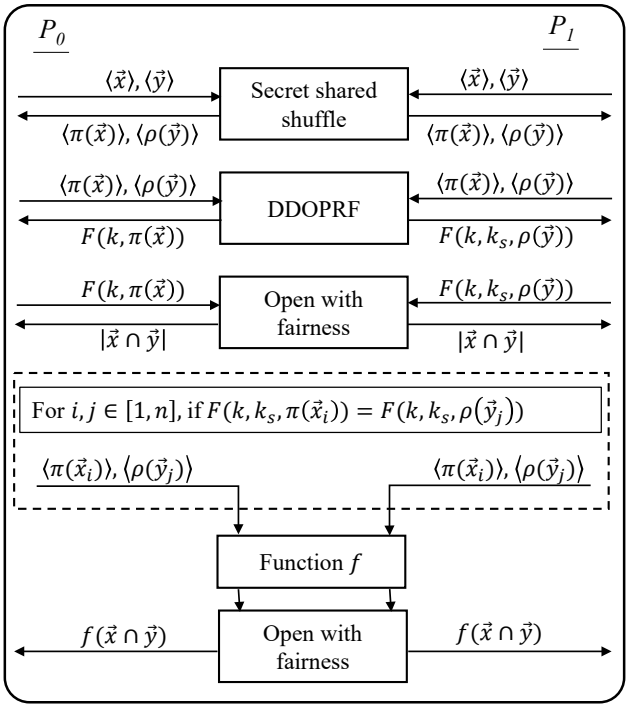

Fig. 3. An overview of Π_{mcPSI}.

A. Workflow Overview

We introduce the high-level workflow of our malicious circuit-PSI protocol Π_{mcPSI}. For two parties P_0 and P_1, their input sets are $\vec{x} = \{x_1, ..., x_n\}$ and $\vec{y} = \{y_1, ..., y_n\}$. As depicted in Fig. 3, we also introduce a "gadget" to guarantee fairness. This "gadget" can be added into one step before any ready-to-opened results.

Initially, P_0 and P_1 will share each item and its authenticated share in the input set, and then P_0 and P_1 both obtain $\langle \vec{x} \rangle$ and $\langle \vec{y} \rangle$. Next, parties will take their authenticated shares as the input of the functionality $\mathcal{F}_{\mathsf{SSS}}$ to get the shuffled shares $\langle \pi(\vec{x}) \rangle$ and $\langle \rho(\vec{y}) \rangle$. The permutations π and ρ are random and not known by any party. In the following, we construct a Distributed Dual-key OPRF protocol Π_{DDOPRF} as a sub-protocol. Π_{DDOPRF} will take the secret-shared shuffled sets as inputs and generate the pseudorandom values for each item in the input sets without revealing the secret keys and shared values. Therefore, P_0 and P_1 will learn the pseudorandom values of the permuted input set $F(k, \pi(\vec{x}))$ and $F(k, k_s, \rho(\vec{y}))$, respectively. Two keys k, k_s are constructed to make sure that the pseudorandom values can be opened with fairness. Then, P_0 and P_1 will invoke a bit decomposition operation [2] to recover the secondary key k_s bit by bit to compare those pseudorandom values $F(k, k_s, \pi(\vec{x}))$ and $F(k, k_s, \rho(\vec{y}))$ and find the equal ones. For $i \in [1, n], j \in [1, n]$, if $F(k, k_s, \pi(\vec{x_i})) = F(k, k_s, \rho(\vec{y_j}))$, then P_0 and P_1 will take the corresponding shares $\langle \pi(\vec{x_i}) \rangle$ and $\langle \rho(\vec{y_j}) \rangle$ for the next

functionality that predefined by two parties. Before parties open the final results, they also use the "gadget" to open the results with fairness. They will choose a secret shared key and use it to encrypt the final results in the secret sharing way. Next, they use a bit decomposition protocol to change the secret shared key bit-wise and open it bit by bit. Finally, P_0 and P_1 can decrypt and obtain the final results $f(x \cap y)$.

Intuition: If two parties use a plain OPRF protocol to compute the PRF results of their input sets and open those results, one dishonest party may abort at any time during the first open process. Though the dishonest party can not learn another party's input items, the adversary may learn how many items in the intersection set have been opened, i.e., $\leq |\vec{x} \cap \vec{y}|$. Then the simulator cannot simulate this adversary behavior, as it cannot learn when the adversary will abort. Therefore, we need to ensure that the adversary will either learn all or nothing, as will the other party. The details will be introduced in the following sections.

B. DDOPRF Protocol from DY-PRF

We propose a Distributed Dual-key OPRF (DDOPRF) protocol based on DY-PRF. In particular, the protocol starts with parties sharing a PRF key $[\![k]\!]$, a secondary key k_s and an input $[\![x]\!]$. At the end of the protocol, the parties output $F(k, x)$ or $F(k, k_s, x)$.

A semi-honest DDOPRF protocol. We design a semi-honest DDOPRF protocol with one PRF key as follows:

- The parties generate a sharing $[\![r]\!]$ for a random secret $r \in \mathbb{Z}_p$ and a beaver triple $([\![a]\!], [\![b]\!], [\![c]\!])$.
- The parties compute $[\![d]\!] \leftarrow [\![r]\!] \cdot ([\![k]\!] + [\![x]\!])$ using $([\![a]\!], [\![b]\!], [\![c]\!])$. The parties open $[\![d]\!]$ to obtain d.
- The parties compute $[\![e]\!] \leftarrow [\![r]\!] \cdot d^{-1}$.
- The parteis locally run $(\![e]\!) \leftarrow \mathsf{Convert}([\![e]\!])$. The parties open $(\![e]\!)$.

The correctness of the above protocol is easy to check:

$$\begin{aligned}[\![e]\!] &= [\![r]\!] \cdot d^{-1} = [\![r]\!] \cdot (r \cdot (k+x))^{-1} \\ &= [\![r \cdot (r \cdot (k+x))^{-1}]\!] \\ &= [\![(k+x)^{-1}]\!]\end{aligned}$$

From the definition of Convert, for two parties, P_0 can compute $g^{[\![e]\!]_0}$ and P_1 can compute $g^{[\![e]\!]_1}$. Clearly,

$$g^{[\![e]\!]_0} \cdot g^{[\![e]\!]_1} = g^e = g^{\frac{1}{k+x}}.$$

Efficiency properties. The above semi-honest DDOPRF protocol has nice efficiency properties. In particular, the parties only perform one secret-shared multiplication for computation $[\![d]\!]$ followed by two openings: one is for opening d and another for opening g^e. It only requires two rounds to compute the OPRF output. As we discussed in related work, previous constructions of OT-based OPRF protocols suffer from diverse shortcomings, including high communication complexity, or not supporting secret-shared data structure.

C. Compile DDOPRF with Malicious Security

This semi-honest DDOPRF enjoys low communication costs. In this section, we will show how to compile the semi-honest protocol with malicious security. While generic techniques (*e.g.*, zero-knowledge proof, GMW compiler) can adapt semi-honest protocols to be malicious secure, the main drawback is their inefficiency. In this section, we show how to adapt the semi-honest DDOPRF with very low overhead.

In the following, we first introduce a *multiplication secret-sharing (MSS)* over $\mathbb{G}$, which resembles LSS over $\mathbb{F}_p$. Then we design *authenticated multiplication secret sharing (AMSS)* over $\mathbb{G}$, borrowing authentication mechanisms from SPDZ-like ASS over $\mathbb{F}_p$. By carefully combining AMSS with ASS, we design maliciously secure DDOPEF with low overhead.

Multiplicative secret sharing over $\mathbb{G}$. Let $\mathbb{G}$ be a prime-order cyclic group with order p, where g is the group generator. We use $(\![x]\!)$ to denote a multiplicative secret sharing (MSS) over $\mathbb{G}$, where P_i holds a share $(\![x]\!)_i$ such that $\prod_{i\in[n]} (\![x]\!)_i = g^x$. Namely, the parties share a secret in the exponent. The above multiplicative secret sharing over $\mathbb{G}$ supports the following computation:

- $(\![z]\!) \leftarrow (\![x]\!) \cdot (\![y]\!)$: P_i computes $(\![z]\!)_i \leftarrow (\![x]\!)_i \cdot (\![y]\!)_i$;
- $(\![z]\!) \leftarrow (\![x]\!)^c$: Given a public $c \in \mathbb{Z}_p$, P_i computes $(\![z]\!)_i \leftarrow (\![x]\!)_i^c$,

where we can verify that $(\![x+y]\!) = (\![x]\!) \cdot (\![y]\!)$ and $(\![c \cdot x]\!) = (\![x]\!)^c$. Namely, multiplication between $(\![x]\!)$ and $(\![y]\!)$ corresponds to addition in the exponent, and $(\![x]\!)^c$ corresponds to scalar multiplication in the exponent.

Authenticated multiplicative secret sharing over $\mathbb{G}$. Similarly, we define authenticated multiplicative secret sharing (AMSS) $\langle\!\langle x \rangle\!\rangle = ((\![x]\!), (\![\gamma(x)]\!))$ over $\mathbb{G}$, where $\gamma(x) = \xi \cdot x \pmod p$. We assume the parties share the MAC key ξ using an LSS sharing $[\![\xi]\!]$. AMSS supports the following local computation:

- $\langle\!\langle z \rangle\!\rangle \leftarrow \langle\!\langle x \rangle\!\rangle \cdot \langle\!\langle y \rangle\!\rangle$: $\langle\!\langle z \rangle\!\rangle \leftarrow ((\![x]\!) \cdot (\![y]\!), (\![\gamma(x)]\!) \cdot (\![\gamma(y)]\!))$,
- $\langle\!\langle z \rangle\!\rangle \leftarrow \langle\!\langle x \rangle\!\rangle^c$: $\langle\!\langle z \rangle\!\rangle \leftarrow ((\![x]\!)^c, (\![c \cdot \gamma(x)]\!))$; here $c \in \mathbb{Z}_p$.

We can verify that $\langle\!\langle x+y \rangle\!\rangle = \langle\!\langle x \rangle\!\rangle \cdot \langle\!\langle y \rangle\!\rangle$ and $\langle\!\langle c \cdot x \rangle\!\rangle = \langle\!\langle x \rangle\!\rangle^c$.

Functionality $\mathcal{F}_{\mathsf{ABB}+}$

Parameters: a prime p; a cyclic group $\mathbb{G}$ of order p, where g is the generator of $\mathbb{G}$.

The ABB functionality contains the following commands:

- $\mathsf{Rand}, \mathsf{RandMul}, \mathsf{Mul}, \mathsf{Open}$ defined as in $\mathcal{F}_{\mathsf{ABB}}$.
- $\langle\!\langle x \rangle\!\rangle \leftarrow \mathsf{Convert}(\langle x \rangle)$: On input a ASS sharing $\langle x \rangle$, output an AMSS sharing $\langle\!\langle x \rangle\!\rangle$.
- $g^x \leftarrow \mathsf{Open}(\langle\!\langle x \rangle\!\rangle)$: On input a group ASS sharing $\langle\!\langle x \rangle\!\rangle$, output g^x to all the parties, and check the MAC value of g^x.

Fig. 4. The extended arithmetic black-box functionality.

Sharing conversion from $\langle x \rangle$ to $\langle\!\langle x \rangle\!\rangle$. We note that an MSS sharing $(\![x]\!)$ over $\mathbb{Z}_p$ can be non-interactively converted from an LSS sharing $[\![x]\!]$ over $\mathbb{G}$ of order p. In particular, each party locally computes $(\![x]\!)_i \leftarrow g^{[\![x]\!]_i}$. Similarly, the

parties can obtain an AMSS sharing $\langle\!\langle x \rangle\!\rangle$ over $\mathbb{G}$ from an ASS sharing $\langle x \rangle$ over $\mathbb{Z}_p$, by simply running the above conversion for $[\![x]\!]$ and $[\![\gamma(x)]\!]$, respectively. In this paper, we use $\langle\!\langle x \rangle\!\rangle \leftarrow \mathsf{Convert}(\langle x \rangle)$ to denote the conversion.

Secretly open g^x from $\langle\!\langle x \rangle\!\rangle$. To open g^x from $\langle\!\langle x \rangle\!\rangle$ correctly, similar to the trick used in ASS, the parties can leverage the MAC sharing $(\!|\gamma(x)|\!)$ to detect any possible error. In particular, the parties run the following open protocol $\mathsf{Open}(\langle\!\langle x \rangle\!\rangle)$ to detect possible errors during opening:

1. Each party P_i reveals its share $(\!|x|\!)_i$. By combining all parties' shares, the parties obtain $g^{x'}$, and $g^{x'}$ may not equate to g^x due to additive errors.
2. Each parties P_i computes $d_i \leftarrow (g^{x'})^{[\![\gamma]\!]_i} / (\!|\gamma(x)|\!)_i$.
3. After each party committing to d_i, all the parties decommit d_i and check whether $\prod_i d_i = 1$ over $\mathbb{G}$. Abort if the check fails.

The above check resembles the MAC check for SPDZ ASS in equation 1, despite the check being evaluated in the exponent. Correctness is easy to check:

$$\begin{aligned}\prod_i d_i &= \prod_i (g^{x'})^{[\![\xi]\!]_i} / \prod_i (\!|\gamma(x)|\!)_i \\ &= g^{\sum_i (x' [\![\xi]\!]_i)} / g^{\gamma(x)} \\ &= g^{x'\xi} / g^{x \cdot \xi},\end{aligned}$$

which equals to 1 over $\mathbb{G}$ *iff* $x = x'$.

The enhanced ABB+ functionality $\mathcal{F}_{\mathsf{ABB+}}$. We formalize an enhanced ABB functionality called ABB+, which captures not only commands for SPDZ ASS over $\mathbb{F}_p$ but also the commands for AMSS over $\mathbb{G}$; here $|\mathbb{G}| = p$. In the following, whenever we require operation over ASS and AMSS authenticated sharings, we will directly resort to this ABB+ functionality. This modular formalization enables a clear and easy-understanding design.

Protocol Π_{DDOPRF}

Parameters: The DY-PRF $F(k, x) = g^{\frac{1}{k+x}}$; an ASS sharing $\langle k \rangle$ for the PRF secret key k; an optional input ASS sharing $\langle k_s \rangle$ for the secondary PRF secret key k_s;

Protocol: On input $\langle x \rangle$ and $\langle k \rangle, \langle k_s \rangle$, do the following:

1. $\langle r \rangle \leftarrow \mathcal{F}_{\mathsf{ABB}}.\mathsf{Rand}()$
2. $\langle d \rangle \leftarrow \langle r \rangle \cdot (\langle k \rangle + \langle x \rangle)$
3. open $d \leftarrow \mathcal{F}_{\mathsf{ABB+}}.\mathsf{Open}(\langle d \rangle)$
4. $\langle e \rangle \leftarrow d^{-1} \cdot \langle r \rangle$
5. If $\langle k_s \rangle$ is provided as an input, $\langle e \rangle \leftarrow \langle e \rangle \cdot \langle k_s \rangle$
6. $\langle\!\langle e \rangle\!\rangle \leftarrow \mathcal{F}_{\mathsf{ABB+}}.\mathsf{Convert}(\langle e \rangle)$
7. open $g^e \leftarrow \mathcal{F}_{\mathsf{ABB+}}.\mathsf{Open}(\langle\!\langle e \rangle\!\rangle)$, e can be $g^{\frac{1}{k+x}}$ or $g^{\frac{k_s}{k+x}}$.

Fig. 5. The malicious DDOPRF protocol.

Our malicious-secure DDOPRF protocol. Using $\mathcal{F}_{\mathsf{ABB+}}$, it is very clear to design a maliciously secure DDOPRF protocol in a modular fashion. The idea is to authenticate the secure computation using prior authenticated mechanisms from ASS and AMSS. We give the theorem of our DDOPRF protocol as follows.

Theorem 1. *In the $\mathcal{F}_{\mathsf{ABB}}$, $\mathcal{F}_{\mathsf{ABB+}}$-hybrid model, the protocol Π_{DDOPRF} implements $\mathcal{F}_{\mathsf{DDOPRF}}$ correctly and securely against malicious adversary.*

The ideal functionality and full proof of Theorem 1 are in Appendix .C.

D. Obtaining Fairness for Free Using Dual-Key

Specifically, we construct a DDOPRF protocol based on the DY-PRF structure to guarantee *fairness* of the following PSI-related computation. The first key is to randomize the input value x and get a PRF value of x, and the secondary key is to re-randomize the PRF value. We give a formal description of our DDOPRF protocol in Fig 5. The ASS sharing $\langle k \rangle$ of the PRF secret key k is generated for computing the pseudorandom value of the input x. In steps 1-4, the parties can compute an ASS sharing $\langle e \rangle = \langle (k+x)^{-1} \rangle$. Instead of directly converting it to AMSS sharing, if parties add a secondary PRF secret key k_s on the PRF value e as in step 5, we can get a PRF value associated with two keys and convert it to AMSS sharing $\langle\!\langle e \rangle\!\rangle = \langle\!\langle g^{\frac{k_s}{k+x}} \rangle\!\rangle$ in step 6. If not, the parties will directly convert e into AMSS sharing. Finally, parties open the final OPRF value in step 7. If a secondary key is used, the OPRF result will be $g^{\frac{k_s}{k+x}}$, otherwise, it opens $g^{\frac{1}{k+x}}$. We will present how our proposed DDOPRF with correlated keys can guarantee *fairness* of Π_{mcPSI} in Section. IV-E.

In conclusion, our protocol Π_{DDOPRF} features a low-round property and low communication. Specifically, Π_{DDOPRF} only requires three rounds of communication: The first round is from computing $\langle e \rangle$ using a beaver triple, the second round opens $\langle\!\langle e \rangle\!\rangle$, and the third round checks the opened result g^e.

E. Our Circuit-PSI from DDOPRF

In this section, we show the construction of a fair and malicious circuit-PSI protocol, which uses Π_{DDOPRF} as a core building block. Besides, we illustrate some other common PSI computations based on circuit-PSI protocol, including PSI with payload computation. Specifically, we present an efficiency enhancement method for malicious secret shared shuffle protocol. We will discuss the details in the following.

Π_{mcPSI} is shown as Fig 6. In the two-party setting, two parties P_0 and P_1 have two input sets $\vec{x}$ and $\vec{y}$, respectively. If two parties need to compute PSI rather than circuit-PSI, they can reveal the set intersection before they send the secret shares into the functionality computation. So they do need to invoke the secret shared shuffle protocol to shuffle their input sets. We will introduce the PSI and circuit-PSI protocols together in more detail in the following.

As shown in step 1, on the input sets $\vec{x}$ and $\vec{y}$, P_0 and P_1 invoke the functionality $\mathcal{F}_{\mathsf{ABB}}.\mathsf{Input}$ to get the ASS shares $\langle \vec{x} \rangle$ and $\langle \vec{y} \rangle$ of their input sets. Next, P_0 and P_1 will use a malicious-secure SSS protocol to secretly shuffle their input shares $\langle \vec{x} \rangle$ and $\langle \vec{y} \rangle$, i.e., $\langle \vec{x'} \rangle = \pi(\langle \vec{x} \rangle) \leftarrow \mathcal{F}_{\mathsf{SSS}}(\langle \vec{x} \rangle)$, $\langle \vec{y'} \rangle = \rho(\langle \vec{y} \rangle) \leftarrow \mathcal{F}_{\mathsf{SSS}}(\langle \vec{y} \rangle)$. Neither P_0 or P_1 know the permutation

Protocol Π_{mcPSI}

Parameters: Two parties P_b $(b \in \{0,1\})$; $\vec{x} = \{x_1, \cdot, x_n\}$ and $\vec{y} = \{y_1, \cdot, y_n\}$ denotes two sets with n values; an authenticated vector $\vec{x}$ where $\vec{x} \in \mathbb{Z}_p^n$; the length of each item in $\vec{x}$ and $\vec{y}$ is ℓ; an ASS sharing $\langle k \rangle$ for the PRF secret key.

Protocol:

1. For $i \in [1,n]$: The parties generate ASS shares of their inputs $\langle x_i \rangle \leftarrow \mathcal{F}_{\mathsf{ABB}}.\mathsf{Input}(x_i)$, $\langle y_i \rangle \leftarrow \mathcal{F}_{\mathsf{ABB}}.\mathsf{Input}(y_i)$;
2. For $f(\vec{x} \cap \vec{y})$: The parties use the SSS protocol to shuffle their shares $\langle \vec{x'} \rangle \leftarrow \mathcal{F}_{\mathsf{SSS}}(\langle \vec{x} \rangle)$, $\langle \vec{y'} \rangle \leftarrow \mathcal{F}_{\mathsf{SSS}}(\langle \vec{y} \rangle)$;
3. Let $\langle k \rangle, \langle k_s \rangle \leftarrow \mathcal{F}_{\mathsf{ABB}^+}.\mathsf{Rand}()$ be the PRF key sharing for Π_{DDOPRF}.
4. Run DDOPRF protocol over $\langle \vec{x'} \rangle$ using key sharing $\langle k \rangle$. Denote the output as $F(k, \vec{x'})$, where each $F(k, x'_i) = g^{\frac{1}{k+x'_i}}$.
5. Run DDOPRF protocol over $\langle \vec{y'} \rangle$ using key sharing $\langle k \rangle, \langle k_s \rangle$. Denote the output as $F(k, k_s, \vec{y'})$,where each $F(k, k_s, y'_i) = g^{\frac{k_s}{k+y'_i}}$.
6. Use the bit decomposition operation over $\langle k_s \rangle$, and get the sharing of sequence $\langle b_1 \cdot 2^{\ell-1} \rangle \cdots \langle b_\ell \cdot 2^0 \rangle$, where b_t $(t \in [1,\ell])$ is the t-th bit of k_s (left most first).
7. The parties open $\langle b_1 \cdot 2^{\ell-1} \rangle \cdots \langle b_\ell \cdot 2^0 \rangle$ one by one, and reconstruct k_s locally;
8. The parties locally compute $(F(k, x'_i))^{k_s}$ to get $F(k, k_s, x'_i)$;
9. $P_{b, b\in\{0,1\}}$ prepares two empty sets $\vec{R}_{X_b}$ and $\vec{R}_{Y_b}$;
10. For $i \in [1,n], j \in [1,n]$:
    a) The parties compare $F(k, k_s, \vec{x'}_i)$ and $F(k, k_s, \vec{y'}_j)$;
    b) If $F(k, k_s, \vec{x'}_i) = F(k, k_s, \vec{y'}_j)$, $P_{b, b\in\{0,1\}}$ picks out the matched record $\vec{R}_{X_b} = \langle \vec{x'}_i \rangle \cup \vec{R}_{X_b}$ and $\vec{R}_{Y_b} = \langle \vec{y'}_j \rangle \cup \vec{R}_{Y_b}$;
11. The parties have learned the number of matched records $|\vec{x'} \cap \vec{y'}|$ and picked out the matched records from $\langle \vec{x'} \rangle$ and $\langle \vec{y'} \rangle$ to perform the following secure computation;
12. For $\vec{x} \cap \vec{y}$: The parties open their shares $\vec{R}_{X_0}$ and $\vec{R}_{Y_0}$ with fairness to recover the intersection set $\{\vec{x} \cap \vec{y}\}$;
13. For $f(\vec{x} \cap \vec{y})$: The parties take their shares $\vec{R}_{X_0}$ and $\vec{R}_{Y_0}$, $\vec{R}_{X_1}$ and $\vec{R}_{Y_1}$ as inputs to a circuit for the functionality f.
14. When two parties open the computation results, they will check the corresponding Mac values. If an error happens, the protocol will abort.

Fig. 6. Protocol Π_{mcPSI} using SPDZ-compatible DDOPRF.

methods π and ρ. For better understanding, we describe the functionality $\mathcal{F}_{\mathsf{SSS}}$ in a black-box way here and will discuss an optimized SSS protocol in detail in the following subsections.

After both parties shuffle their input sharings, they can not learn anything from the shuffled sharings $\vec{x'}$ and $\vec{y'}$. Then, in steps 3-5, parties will invoke the DDOPRF protocol Π_{DDOPRF} as defined in Fig. 5 to generate the pseudorandom values for the permuted sharing. Specifically, let $\langle k \rangle, \langle k_s \rangle \leftarrow \mathcal{F}_{\mathsf{ABB}^+}.\mathsf{Rand}()$ be the PRF keys sharings for DDOPRF protocol. Two parties run DDOPRF protocol over $\langle \vec{x'} \rangle$ and $\langle \vec{y'} \rangle$ using key sharings $\langle k \rangle$ and $\langle k_s \rangle$. In step 4, two parties output the OPRF values of $\vec{x'}$ under the PRF key sharing $\langle k \rangle$, denoted as $\{F(k, \vec{x'}_i)\}_{i\in[1,n]}$. In step 5, two parties output the OPRF values of $\vec{y'}$ under the PRF key sharing $\langle k \rangle$ and secondary key sharing $\langle k_s \rangle$, denoted as $\{F(k, k_s, \vec{y'}_i)\}_{i\in[1,n]}$. To make sure that two parties can compare the OPRF values at the same time, and they can get all comparison results or nothing, two parties will run a bit decomposition protocol on the secondary key sharing $\langle k_s \rangle$ in step 6. Then, they can open the bit composition of k_s one by one to reconstruct k_s. After that, two parties can compute $(F(k, x'_i))^{k_s} = g^{\frac{k_s}{k+x'_i}}$ with $g^{\frac{k_s}{k+y'_i}}$ to get which items are in the intersection set.

In the following, the parties intend to find the corresponding sharings in the intersection set. In step 9, the parties will prepare two empty sets to store the sharings. In step 10, the parties can find matches over $\{F(k, k_s, \vec{x'}_i)\}_{i\in[1,n]}$ and $\{F(k, k_s, \vec{y'}_j)\}_{j\in[1,n]}$ After parties learn all matched records. They pick out the matched records from $\langle \vec{x'} \rangle$ and $\langle \vec{y'} \rangle$, and perform the following secure computation. Therefore, in step 8, the parties use the stored sharings, whose DDOPRF results match, as the inputs of the following circuit to achieve $f(\vec{x} \cap \vec{y})$. With the help of a malicious secure Π_{DOPRF}, P_0 and P_1 will obtain pseudorandom results $\{F(k, k_s, \vec{x'}_i)\}_{i\in[1,n]}$ and $\{F(k, k_s, \vec{y'}_j)\}_{j\in[1,n]}$. Since the input values have been shuffled before Π_{DDOPRF}, P_0 and P_1 can not correlate their input values $\vec{x}$ and $\vec{y}$ with the pseudorandom results. At the end of Π_{DDOPRF}, parties pick the corresponding shares in the permuted shared sets $\langle \vec{x'} \rangle$ and $\langle \vec{y'} \rangle$, and send those shares to the following circuit. Therefore, this process achieves the computation of $f(\vec{x} \cap \vec{y})$. As we can see, the consistency issue we mentioned before will be solved since the shares sent into the circuit are with MAC shares. They can open the final results to verify whether parties modify the shares before the circuit computation.

F. Fairness of Protocol Π_{mcPSI}

The principle of fairness is defined such that if one party knows certain information, the other party should also be aware of the same information [18]. In Π_{mcPSI}, one party obtains PRF values of the input set $g^{\frac{1}{k+x'_i}}$ $(i \in [1,n])$, and another party gets re-randomized input set with two PRF keys $\langle g^{\frac{k_s}{k+y'_i}} \rangle$. We propose that the secondary key k_s is used to re-randomize the PRF value and ensure the fairness. If two parties only compute standard PRF values with only one PRF key, i.e., $g^{\frac{1}{k+x'_i}}$ and $g^{\frac{1}{k+y'_i}}$, then they open those PRF values to each other one by one to compare each value. However, if one of the parties is corrupted, the corrupted party can quit at any time during the opening process. We can observe that, the corrupted party can learn some information, such as the intersection set having at least t items if the corrupted party finds t items in the intersection before quitting. Moreover, the corrupted party might obtain more information than another party. If the corrupted party aborts after another party shares one item (if this item belongs to the intersection set), the corrupted party can learn the intersection set at least has one item but

another party learns nothing. So it would be hard to measure the leakage based on the ideal/real-world simulation method. Since the simulator can not simulate when the adversary would abort and define the amount of leakage.

Therefore, we construct Π_{DDOPRF}, and we find that if we construct two correlated keys for the PRF value, the problems mentioned above can be solved. Specifically, two parties will open $g^{\frac{1}{k+x'_i}}$ and $\langle g^{\frac{k_s}{k+y'_i}} \rangle$. Because the randomization of the PRF value $\langle g^{\frac{k_s}{k+y'_i}} \rangle$ is guaranteed by the secondary key k_s, two parties can not distinguish it with a random value. Afterward, they invoke a bit decomposition protocol to recover k_s bit by bit. Subsequently, the two parties can locally compute the PRF values for input sets with the same keys to get the intersection result. We can observe that, the adversary will learn the final intersection set or nothing.

If two parties intend to compute fair f(PSI), for the final result of f(PSI), two parties also can use the same trick as used in DDOPRF to guarantee *fairness*. To be specific, before the parties reveal the final shares, they will select a secret key to encrypt their shares. After they open the encrypted shares, they will open the encrypted keys bit-by-bit to decrypt the shares and get the final results.

Malicious PSI with payload computation (labeled circuit-PSI). We also present how to use Π_{mcPSI} to achieve PSI with payload computation as follows. Let us assume two secret-shared tables $\langle X \rangle$ and $\langle Y \rangle$, and X and Y are both two-column tables of P_0 and P_1, where the first column is the ID column and the second is the payload column. The parties want to perform an intersection over two ID columns for X and Y and then select out all the payload values associated with the IDs in the intersection. The parties do not want to reveal which records are used in the computation, and the only allowed leakage is the number of matched records, *i.e.*, the cardinality of the intersection.

Similarly, parties will compute the ASS for their input matrices, shown as $\langle X \rangle$, $\langle Y \rangle$. Next, the parties perform row-wise secret-shared shuffle over $\langle X \rangle$ and $\langle Y \rangle$. Let us denote the shuffled table as $\langle X' \rangle = \langle \pi(X) \rangle$ and $\langle Y' \rangle = \langle \rho(Y) \rangle$ for some random permutation methods π and ρ. Neither of the parties learns about the permutation methods. Then, the parties invoke Π_{DDOPRF}. For Π_{DDOPRF}, they first sample a random sharing $\langle k \rangle$ as the ASS key sharing of the DY-PRF. Parse $\langle X' \rangle$ as $(\langle X'^{(0)} \rangle, \langle X'^{(1)} \rangle)$ and $\langle Y' \rangle$ as $(\langle Y'^{(0)} \rangle, \langle Y'^{(1)} \rangle)$, where $X'^{(0)}$ and $Y'^{(0)}$ are the ID columns. The parties run Π_{DDOPRF} over the ID column of $\langle X' \rangle$ using PRF key sharing $\langle k \rangle$, and run Π_{DDOPRF} over $\langle Y' \rangle$ with PRF key sharing $\langle k \rangle$ and secondary key sharing $\langle k_s \rangle$. Then, they invoke the bit decomposition protocol on $\langle k_s \rangle$ to open this key bit by bit. So parties can compute $\{F(k, k_s, X'^{(0)}_i)\}_{i \in [1,n]}$ to do the following comparison. At the end of this protocol, the parties can learn the pseudorandom values of those ID columns, denoted as $\{F(k, X'^{(0)}_i)\}_{i \in [1,n]}$ and $\{F(k, Y'^{(0)}_i)\}_{i \in [1,n]}$. After that, P_0 prepares two empty tuple sets R_{X_0} and R_{Y_0}, and P_1 prepares R_{X_1} and R_{Y_1}, and they will find all the matched records between $\langle \pi(X) \rangle$ and $\langle \rho(Y) \rangle$ from the DDOPRF output, and put those matched sharings into the empty sets. More concretely, for $i \in [1, n], j \in [1, n]$, the parties compare $F(k, X'^{(0)}_i)$ and $F(k, Y'^{(0)}_j)$. If $F(k, X'^{(0)}_i) = F(k, Y'^{(0)}_j)$, P_0 (P_1) picks out the matched record $R_{X_0} = \langle X'_i \rangle \cup R_{X_0}$ ($R_{X_1} = \langle X'_i \rangle \cup R_{X_1}$) and $R_{Y_0} = \langle Y'_j \rangle \cup R_{Y_0}$ ($R_{Y_1} = \langle Y'_i \rangle \cup R_{Y_1}$). For payload computation, the parties take their shares $R^{(1)}_{X_0}$ and $R^{(1)}_{Y_0}$, $R^{(1)}_{X_1}$ and $R^{(1)}_{Y_1}$ as inputs to the following payload computation. Note that due to the secret-shared shuffle, the parties do not know which records are matched, and they only learn the number of matched records at the end of the protocol.

G. Concrete SSS Protocol Instantiation

Our protocol relies on a $\mathcal{F}_{\mathsf{SSS}}$ in a black-box way, meaning that any malicious secure SSS protocol that secretly computes $\mathcal{F}_{\mathsf{SSS}}$ can be used here. When diving into the concrete instantiations, however, some existing SSS protocols [5, 12, 23] based on permutation networks suffer from high round and communication complexity, which may incur high latency when used in the WAN setting.

We provide a concrete instantiation for a malicious-secure SSS protocol with better efficiency and use it as a black-box way. The used malicious-secure SSS protocol [49] is based on a recently proposed two-party SSS protocol from [12] known as the CGP protocol, which is only secure in the presence of a semi-honest adversary. The details of the malicious SSS protocol [49] and our enhancement are presented in Appendix B.

V. Security Proof

We follow the simulation-based security model [31] with malicious security and static corruption. The security goals are defined as an ideal functionality $\mathcal{F}$. This ideal functionality works as a trusted entity that receives inputs from parties, performs the defined computation, and outputs results to parties. In the real world, an adversary A who represents a corrupted party C will run the protocol with the other honest parties. In the ideal world, a simulator S will interact with $\mathcal{F}$.

Definition 2. *A protocol Π securely computes functionality $\mathcal{F}$ in the presence of a malicious adversary if for every PPT adversary A there exists a PPT simulator S, such that*

$$Real_{\Pi, A(z), C}(1^\kappa, 1^\lambda, x_{i, i \notin C}) \stackrel{c}{\equiv} Ideal_{\Pi, S(z), C}(1^\kappa, 1^\lambda, x_{i, i \notin C}).$$

The left side of the equation represents the joint output from the honest parties and adversity A, and x_i represents the input from a party P_i and z is the auxiliary input from A. Similarly, the right side denotes the joint output from the honest parties and simulator S. We say that Π can securely compute the functionality $\mathcal{F}$ with less than statistical error 2^λ under the malicious model.

Then we define the ideal functionality of $\mathcal{F}_{\text{mcPSI}}$ as in Fig. 7

Theorem 2. *In the $\mathcal{F}_{\mathsf{ABB}}$,$\mathcal{F}_{\mathsf{ABB}^+}$,$\mathcal{F}_{\mathsf{SSS}}$,$\mathcal{F}_{\mathsf{DDOPRF}}$-hybrid model, the protocol Π_{mcPSI} implements $\mathcal{F}_{\text{mcPSI}}$ correctly and securely against malicious adversary, and achieves $(2, 0)$-fair.*

Functionality $\mathcal{F}_{\text{mcPSI}}$

Parameters: The party P_0 inputs $\vec{x} = \{x_1, \cdots, x_n\}$, and another party P_1 has an input set $\vec{y} = \{y_1, \cdots, y_n\}$;

$\mathcal{F}_{\text{mcPSI}}$:

1. On receiving $(\mathcal{F}_{\text{mcPSI}}, \vec{x})$ from P_0 and $(\mathcal{F}_{\text{mcPSI}}, \vec{y})$ from P_1, the functionality stores $\vec{x}$ and $\vec{y}$ and waits. If any party aborts outputs $\perp$ to both parties and aborts. Otherwise, continue.
2. On recceiving (compute) from both parties, the functionality outputs the computation results $f(\vec{x} \cap \vec{y})$ and size of the intersection set $|\vec{x} \cap \vec{y}|$ to both patries if it does not abort. Otherwise, $\perp$ is output to parties.

Fig. 7. Ideal functionality of $\mathcal{F}_{\text{mcPSI}}$.

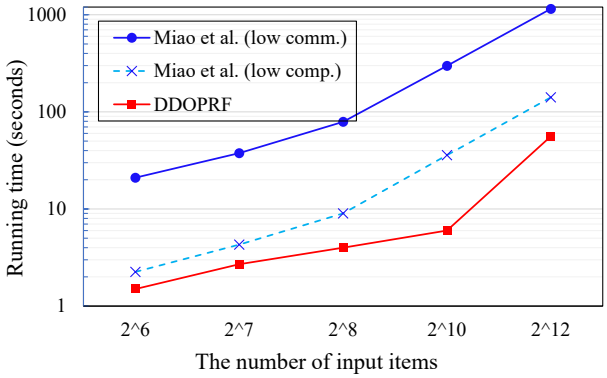

Fig. 8. Running time (s) for computing PRF values of input items.

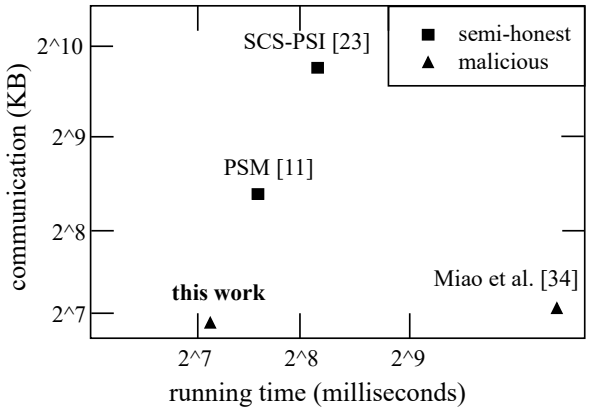

Fig. 9. Time and communication for circuit-PSI protocols on $n = 256$ and LAN network setting.

Proof Sketch. In this part, we give an essential proof sketch of $\mathcal{F}_{\text{mcPSI}}$ to establish that it is maliciously secure and fair. First, we solve the consistent issue by designing an SPDZ-compatible OPRF protocol (i.e., DDOPRF). That is, we augment OPRF by adding the authentication mechanisms provided by SPDZ in the secret-sharing format. Instead of using heavy asymmetric-based commitment schemes, SPDZ provides the symmetric-key counterpart "message authentication codes" (MAC) for authentication. A MAC is a way of authenticating a value, ensuring that any revisions to the value can be detected by checking its MAC. DDOPRF perfectly integrates all the features in SPDZ, including MAC. Therefore, the computation results sent to a functionality f will be checked by MAC, and any changes to those results will be detected. At the end of this protocol, the correctness of $\mathcal{F}_{\text{mcPSI}}$ can be ensured.

Next, we give a discussion about the fairness of our protocol. $\mathcal{F}_{\text{mcPSI}}$ achieves $(2, 0)$-fair. According to the partial fairness definition in Definition. 1, whenever the adversary aborts the protocol, the upper bound of its advantage in recovering the results is known one more bit than the other party. Therefore, the recovering time of the adversary will be half of the other party, and it can achieve the same probability of success. Besides, the fairness of $\mathcal{F}_{\text{mcPSI}}$ is guaranteed by the bit-decomposition protocol, which is also built on SPDZ and secure under the malicious model.

Nevertheless, the correctness, security, and fairness of $\mathcal{F}_{\text{mcPSI}}$ are guaranteed by the primitives used in it. The detailed simulation-based proofs are shown in Appendix .C.

VI. Implementation and Performance

A. Experiment Environment

We implement our protocol in C++ and based on YACL[3], which provides several cryptographic interfaces (e.g., pseudorandom generator, oblivious transfer, network). We run all experiments on a desktop PC equipped with 12th Gen Intel(R) Core(TM) i9-12900K at Ubuntu 20.04 LTS and 125 GB of memory. We run our protocols in two different network settings with the Linux tc command. One is the local-area network (LAN) with 10 GBps. Another is the wide-area network (WAN) with 100 Mbps. In our paper, the computational security parameter is $\kappa = 128$, the statistical security parameter is $\lambda = 64$, and the size of each element is $\ell = 128$.

[3]https://github.com/secretflow/yacl

set size n	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
Offline	0.06	0.58	10.43	90.18	1445.66	23131	-
Online (without fairness)	0.12	0.50	1.83	7.32	28.78	115.90	455.95
Online (with fairness)	0.14	0.58	2.05	8.07	32.03	128.02	512.42
Comm.	0.08	0.31	1.15	4.52	17.98	71.85	287.34

TABLE I
Running time (in s) and Communication cost (in MB) of Π_{mcPSI} for different set sizes ($n \in \{2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}\}$) in LAN setting.

B. Performance of DDOPRF

We first report the performance of our DDOPRF protocol in terms of the running time and communication cost. The performance is measured for generating PRF values for items in the input sets from two parties, where the number of items n is taken from $\{2^6, 2^7, 2^8, 2^{10}, 2^{12}\}$. Besides, we compare our DDOPRF protocol with the OPRF protocol used in [34]. Fig. 8 shows the running time of our DDOPRF protocol with the OPRF protocol used in Miao et al. [34]. In Miao's work, they also propose an OPRF protocol based on DY-PRF and

set size n		2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
LAN								
End2End Time	Ours	**0.14**	0.57	2.08	8.05	32.18	128.16	511.24
	PSM [11]	0.219	0.334	0.545	0.70	1.65	6.07	24.78
		$\approx 1.6\times$						
Comm.	Ours	**0.08**	**0.31**	**1.15**	**4.52**	**17.98**	**71.85**	**287.34**
	PSM [11]	0.38	1.53	6.10	24.33	99.48	397.65	1700.82
		$\approx 4.8\times$	$\approx 4.9\times$	$\approx 5.3\times$	$\approx 5.4\times$	$\approx 5.5\times$	$\approx 5.5\times$	$\approx 5.9\times$
WAN (100Mbps)								
End2End Time	Ours	**0.19**	**0.73**	**2.40**	9.07	35.16	139.39	558.89
	PSM [11]	1.779	2.526	4.092	7.73	16.49	42.85	162.61
		$\approx 9.4\times$	$\approx 3.5\times$	$\approx 1.7\times$				-

TABLE II
Running time (in seconds) and Communication cost (in MB) of Π_{mcPSI} and the circuit-PSI protocol in [11] for different set sizes ($n \in \{2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}\}$) in different network settings.

set size n	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
Ours	0.14	0.58	2.05	8.07	32.03	128.02	512.42
SCS-PSI [23]	0.309	0.81	3.464	14.87	63.86	274.12	-
Miao et al. [34]	8.25	33	141	553.8	2215	8860	25583

TABLE III
Runtime (in seconds) of Π_{mcPSI} and PSI protocols in [23, 34] for different set sizes ($n \in \{2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}\}$)

combine it with distributed keys. As we can see from the results, the running time of DDOPRF is around 100x faster than that of Miao's protocol (low comm.), and 10x faster than Miao's optimized OPRF protocol (low comp.). Specifically, the running time of DDOPRF includes the offline and online time. However, it does not conclude the dual-key computation part. That is to say, the running time of DDOPRF shown in Fig. 8 does not guarantee fairness, since Miao's work also does not consider fairness property. Because we test the malicious DDOPRF protocol here, the malicious OTs protocol from [48] is used as a primitive in DDOPRF. We note that all OTs execution and the MAC check can be done in a batch in one round. Besides, we also use VOLE to enhance the generation of MAC on SPDZ, and this technique was proposed in [52].

set size n	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
Ours	0.08	0.31	1.15	4.52	17.98	71.85	287.34
SCS-PSI [23]	8.76	42.41	207.20	1002.62	4941.83	24139	-
Miao et al. [34]	0.12	0.481	1.89	7.1	28.3	111.22	436.720

TABLE IV
Communication cost (in MB) of Π_{mcPSI} and PSI protocols in [40, 47] for different set sizes ($n \in \{2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}\}$).

C. Performance of Malicious Circuit-PSI

In this section, we show the thorough performance of our malicious circuit-PSI protocol. Our malicious circuit-PSI protocol includes three parts, 1) secret shared shuffle, 2) DDOPRF, and 3) functionality f computation (with fairness).

We first give the specific numbers of Π_{mcPSI} in the localhost setting in Table. I. In this table, we divide the running time of Π_{mcPSI} into two parts: offline and online times. And we also represent the online time with/without fairness guaranteed. During the offline process, it generates quantities of beaver triples and correlated random values for later online shuffle and DDOPRF protocol. As for fairness, to guarantee the fairness in DDOPRF protocol, we need to change the arithmetic sharings of a key to boolean sharings vis bit decomposition protocol. In our implementation, we use $0/1$ arithmetic sharings to substitute the boolean sharings to avoid complex implementation. When the set size n equals 2^{20}, it takes hours and the socket will be dead so we do not give a specific number. However, we could still infer the time since the offline communication cost is $O(nlogn)$ and computation cost is $O(n^2)$. The primary expense incurred during offline periods stems from the shuffle process.

Then, we compare Π_{mcPSI} with the existing two-party PSI protocols [23, 34]. To compare our protocol to other PSI protocols [23, 34], Π_{mcPSI} does not include the offline running time. For the circuit computation, we use the same functionality as used in [34], which aims to compute the sum of items in the intersection set. And we both focus on the malicious setting. Nevertheless, Huang's PSI protocol [23] brings a semi-honest PSI based on garbled circuit methods. Their work also can be trivially extended to achieve the PSI-sum computation and be secure under the malicious setting. Therefore, we compare our protocol with those representative works as presented in Table. III. It is worth mentioning that, the running time of Π_{mcPSI} does not include offline time, and it includes the online time of SSS, DDOPRF (with fairness), and sum computation. As we can conclude from Table. III, the online time is similar to the running time of DDOPRF protocol, since the online time of SSS is nearly 0 when the size of input set is rather small ($n < 2^{12}$). Compared to Miao's work, the running time of Π_{mcPSI} is nearly 50x faster. Compared to the semi-honest PSI [23], our work is around 1.9x more efficient in the running time.

Next, we also compare the communication costs of Π_{mcPSI} with [23, 34] shown in Table. IV. Notably, we only count the online communication cost of Π_{mcPSI}. We also present a

Protocol	Aims	Fairness	Comm. asymptotic	Assumption
Semi-Honest Security				
SCS-PSI [23]	PSI	✗	$(2\ell nlog(2n) + ((3n-1)\ell - n) + 2\ell nlog^2(2\hat{n}))\phi$	CDH
PSM [11]	PSI, circuit-PSI	✗	$0.25\ell n\lambda + 0.5\ell\lambda + 8\ell n$	
VOLE-PSI [47]	PSI, circuit-PSI	✗	$(\lambda + 2log(n))n + 2^{17}\kappa n^{0.05} + \kappa n +$ |baseOT|	LPN+CDH
Malicious Security				
PaXoS [40]	PSI	✗	$2\kappa n + \ell(2.4n + 2\lambda + \chi) + \lambda(2.4n + 2\ell) +$ |baseOT|	CDH
VOLE-PSI [47]	PSI	✗	$3\kappa n + 2^{17}\kappa n^{0.05} +$ |baseOT|	LPN+CDH
Blazing [44]	PSI	✗	$2.3\kappa n + 2^{14.5}\kappa +$ |baseOT|	LPN+CDH
Dong et al. [18]	PSI	✓ need a semi-trusted server	$O(n^2)$	CDH
Miao et al. [34]	PSI-Sum	✗	$O(n)$	
Ours	PSI, circuit-PSI	✓	$(2\lambda + 2)\ell n$	

TABLE V
Theoretical comparison of different PSI-related protocols, using the computational security $\kappa = 128$, the length of each item ℓ, and the statistical security $\lambda = 40$. n is the size of input set, and we consider the sizes of two input sets to be equal.

comparison of Π_{mcPSI} with other PSI-related works on $n = 2^8$ in Fig. 9. We can conclude from those tables that Π_{mcPSI} features good communication performance, which is several orders of magnitude faster than Huang's protocol, and around 1.5x less than the communication cost of Miao's protocol.

Next, we show the running time and communication cost of Π_{mcPSI} in different network settings and compare it with one of the state-of-the-art semi-honest circuit-PSI protocol [11] in Table. II. Our communication and computation costs are linear than the size of input sets. Compared to the semi-honest circuit-PSI protocol [11], we can conclude that Π_{mcPSI} achieves better online efficiency towards the smaller size of input sets in the WAN setting. Specifically, all the online times include the computation for fairness. Besides, our protocol has better online communication cost than [11]. As we can conclude from the results, our malicious circuit-PSI protocol brings some extra costs at certain levels compared to state-of-the-art circuit-PSI protocol [11], but our efficiency remains competitive in terms of communication costs and running time on the WAN network.

D. Theoretical Analysis

In Table VI-B, we provide a thorough theoretical comparison of our protocols with other PSI-related semi-honest and malicious protocols in different security settings. SCS-PSI [23] is a pure circuit-based PSI protocol in the semi-honest setting. Its communication cost is linear with the number of the used gates. Therefore, we use ϕ to represent the communication cost for one non-free gate. Next, we show the main communication cost of PSM [11], a private membership test protocol.

As for the malicious protocols, although Miao's work achieves linear communication complexity, it relies on too many asymmetric operations, including the Pedersen commitments and ElGamal encryptions, resulting in low efficiency (shown in Tables III and IV). In Π_{mcPSI}, we consider $\ell = 2\kappa$ since the field size of the DDOPRF is 2κ. More specifically, in the shuffle protocol, the offline communication cost is $O(\kappa n \log n + \kappa n)$. The online communication cost of the shuffle process is $2\lambda n$ as it needs to shuffle the input items and their MAC values. Then, in the DDOPRF protocol, the communication cost is $2\lambda\ell n$. In Table. VI-B, we focus solely on the online linear communication cost of Π_{mcPSI}. In conclusion, Π_{mcPSI} is not only the first malicious circuit-PSI protocol but also achieves fairness and better efficiency.

VII. Conclusion and Discussion

This paper proposes a malicious secure and fair circuit-PSI protocol. Specifically, we design a distributed dual-key oblivious PRF, which can be used for designing our circuit-PSI protocol. Besides, we also bring up some gadgets to improve the efficiency of our protocol, including improved malicious secret-shared shuffle method and batched consistency check. And we believe those sub-protocols are of independent interest.

Future work. Many recent researches focus on how to extend the two-party PSI-related protocols to the multi-party setting. If we trivially extend our malicious circuit-PSI protocol into a multi-party setting, it needs a multi-party secret-shared shuffle protocol, and the DDOPRF protocol will be executed between n parties. It may result in low efficiency and non-linear communication complexity. Therefore, it remains to design a maliciously secure circuit-PSI protocol with better complexity.

References

[1] Aydin Abadi and Steven J Murdoch. Earn while you reveal: Private set intersection that rewards participants. *arXiv preprint arXiv:2301.03889*, 2023.

[2] Toshinori Araki, Assi Barak, Jun Furukawa, Marcel Keller, Yehuda Lindell, Kazuma Ohara, and Hikaru Tsuchida. Generalizing the spdz compiler for other protocols. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 880–895, 2018.

[3] Gilad Asharov, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Ariel Nof, Benny Pinkas, Katsumi Takahashi, and Junichi

Tomida. Efficient secure three-party sorting with applications to data analysis and heavy hitters. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 125–138, 2022.

[4] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 535–548, 2013.

[5] Borja Balle, James Bell, Adria Gascón, and Kobbi Nissim. Private summation in the multi-message shuffle model. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 657–676, 2020.

[6] Amos Beimel. Secret-sharing schemes: A survey. In *International conference on coding and cryptology*, pages 11–46. Springer, 2011.

[7] Michael Ben-Or, Oded Goldreich, Silvio Micali, and Ronald L Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.

[8] Václad E Beneš. Optimal rearrangeable multistage connecting networks. *Bell system technical journal*, 43(4):1641–1656, 1964.

[9] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Near-optimal oblivious key-value stores for efficient psi, psu and volume-hiding multi-maps. Cryptology ePrint Archive, Paper 2023/903, 2023. https://eprint.iacr.org/2023/903.

[10] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. Sok: Oblivious pseudorandom functions. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 625–646. IEEE, 2022.

[11] Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-psi with linear complexity via relaxed batch opprf. *Proceedings on Privacy Enhancing Technologies*, 2022.

[12] Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. Secret-shared shuffle. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part III 26*, pages 342–372. Springer, 2020.

[13] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious prf. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III 40*, pages 34–63. Springer, 2020.

[14] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 364–369, 1986.

[15] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[16] Sumit Kumar Debnath and Ratna Dutta. New realizations of efficient and secure private set intersection protocols preserving fairness. In *International Conference on Information Security and Cryptology*, pages 254–284. Springer, 2016.

[17] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*, pages 416–431. Springer, 2005.

[18] Changyu Dong, Liqun Chen, Jan Camenisch, and Giovanni Russello. Fair private set intersection with a semi-trusted arbiter. In *Data and Applications Security and Privacy XXVII: 27th Annual IFIP WG 11.3 Conference, DBSec 2013, Newark, NJ, USA, July 15-17, 2013. Proceedings 27*, pages 128–144. Springer, 2013.

[19] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 789–800, 2013.

[20] Thai Duong, Duong Hieu Phan, and Ni Trieu. Catalic: Delegated psi cardinality with applications to contact tracing. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part III 26*, pages 870–899. Springer, 2020.

[21] Saba Eskandarian and Dan Boneh. Clarion: Anonymous communication from multiparty shuffling protocols. *Cryptology ePrint Archive*, 2021.

[22] Badih Ghazi, Noah Golowich, Ravi Kumar, Rasmus Pagh, and Ameya Velingker. On the power of multiple anonymous messages: Frequency estimation and selection in the shuffle model of differential privacy. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 463–488. Springer, 2021.

[23] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[24] 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. IEEE, 2020.

[25] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842, 2016.

[26] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: making spdz great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 158–189. Springer, 2018.

[27] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious prf with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 818–829, 2016.

[28] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1257–1272, 2017.

[29] Peeter Laud. Linear-time oblivious permutations for spdz. In *Cryptology and Network Security: 20th International Conference, CANS 2021, Vienna, Austria, December 13-15, 2021, Proceedings 20*, pages 245–252. Springer, 2021.

[30] Phi Hung Le, Samuel Ranellucci, and S Dov Gordon. Two-party private set intersection with an untrusted third party. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2403–2420, 2019.

[31] Yehuda Lindell. How to simulate it–a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pages 277–346, 2017.

[32] Yang Liu, Yan Kang, Tianyuan Zou, Yanhong Pu, Yuanqin He, Xiaozhou Ye, Ye Ouyang, Ya-Qin Zhang, and Qiang Yang. Vertical federated learning. *arXiv preprint arXiv:2211.12814*, 2022.

[33] Qiyao Luo, Yilei Wang, and Ke Yi. Frequency estimation in the shuffle model with almost a single message. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2219–2232, 2022.

[34] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In *Annual International Cryptology Conference*, pages 3–33. Springer, 2020.

[35] Daniel Morales, Isaac Agudo, and Javier Lopez. Private set intersection: A systematic literature review. *Computer Science Review*, 49:100567, 2023.

[36] Emmanuela Orsini. Efficient, actively secure mpc with a dishonest majority: a survey. In *Arithmetic of Finite Fields: 8th International Workshop, WAIFI 2020, Rennes, France, July 6–8, 2020, Revised Selected and Invited Papers 8*, pages 42–71. Springer, 2021.

[37] M Papoutsidakis, E Symeonaki, D Piromalis, and D Tseles. Modern shipping navigation based on telemetry and communication systems. *International Journal of Computer Applications*, 975:8887, 2017.

[38] Benny Pinkas. Fair secure two-party computation. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 87–105. Springer, 2003.

[39] 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: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pages 401–431. Springer, 2019.

[40] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Psi from paxos: fast, malicious private set intersection. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 739–767. Springer, 2020.

[41] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based psi with linear communication. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38*, pages 122–153. Springer, 2019.

[42] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Transactions on Privacy and Security (TOPS)*, 21(2):1–35, 2018.

[43] Pille Pullonen et al. Actively secure two-party computation: Efficient beaver triple generation. *Instructor*, 2013.

[44] Srinivasan Raghuraman and Peter Rindal. Blazing fast psi from improved okvs and subfield vole. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2505–2517, 2022.

[45] Priyanga Rajaram, Mark Goh, and Jianying Zhou. Guidelines for cyber risk management in shipboard operational technology systems. In *Journal of Physics: Conference Series*, volume 2311, page 012002. IOP Publishing, 2022.

[46] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 235–259. Springer, 2017.

[47] Peter Rindal and Phillipp Schoppmann. Vole-psi: fast oprf and circuit-psi from vector-ole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 901–930. Springer, 2021.

[48] Lawrence Roy. Softspokenot: Communication–computation tradeoffs in ot extension. *Cryptology ePrint Archive*, 2022.

[49] Xiangfu Song, Dong Yin, Jianli Bai, Changyu Dong, and Ee-Chien Chang. Secret-shared shuffle with malicious security. Cryptology ePrint Archive, Paper 2023/1794, 2023. https://eprint.iacr.org/2023/1794.

[50] Abraham Waksman. A permutation network. *Journal of the ACM (JACM)*, 15(1):159–163, 1968.

[51] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 21–37, 2017.

[52] 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. IEEE, 2021.

[53] Yaxi Yang, Yao Tong, Jian Weng, Yufeng Yi, Yandong Zheng, Leo Yu Zhang, and Rongxing Lu. Prirange: Privacy-preserving range-constrained intersection query over genomic data. *IEEE Transactions on Cloud Computing*, 2022.

[54] Yaxi Yang, Jian Weng, Yufeng Yi, Changyu Dong, Leo Yu Zhang, and Jianying Zhou. Predicate private set intersection with linear complexity. In *International Conference on Applied Cryptography and Network Security*, pages 143–166. Springer, 2023.

[55] Andrew Chi-Chih Yao. How to generate and exchange secrets extended abstract. In *27th FOCS*, pages 162–167, 1986.

APPENDIX A
ARITHMETIC BLACK-BOX FUNCTIONALITY

The detailed implementation of the functionality $\mathcal{F}_{\mathsf{ABB}}$ is as follows.

SPDZ-style preprocessing. In the preprocessing phase, the SPDZ-style protocol will use oblivious transfer as an underlying technique for generating triples $\mathsf{RandomMul}()$ [25].

RandomMul: The pre-processing of random multiplication $\mathsf{RandomMul}()$ is to generate multiplication triples. The triple generation protocol is as follows. For two parties P_0 and P_1, each party samples $\vec{a}^{(0/1)}, b^{(0/1)}$, which are randomly sampled from a finite field $\mathbb{F}$. Both parties call the Random OT protocol $\mathcal{F}_{\mathsf{ROT}}^{k\tau,k}$, where each party inputs $\vec{a}^{(0/1)}$ in bit form, i.e., $(a_1^{(0/1)}, ..., a_{\tau k}^{(0/1)}) = \vec{g}^{-1}(a^{(0/1)})$, where $\vec{g} = (1, 2, 2^2, ..., 2^{k-1})$, integer parameter $\tau \geq 3$, k is the length of generated triple value. Then they do the following:

1. P_1 receives $q_{0,h}^{(1,0)}$, $q_{1,h}^{(1,0)}$, and P_0 receives $s_h^{(0,1)} = q_{a_h^{(0)},h}^{(1)}$, for $h = 1, ..., \tau k$;
2. P_1 sends $d_h^{(1,0)} = q_{0,h}^{(1,0)} - q_{1,h}^{(1,0)} + b^{(1)}$;
3. P_0 sets $t_h^{(0,1)} = s_h^{(0,1)} + a^{(0)} \cdot d_h^{(1,0)} = q_{0,h}^{(}1, 0) + a_h^{(0)} \cdot b^{(j)}$, for $h \in [1, \tau k]$. Then sets $q_h^{(1,0)} = q_{0,h}^{(}1, 0)$.
4. Two parties split $(t_1^{(0,1)}, ..., t_{\tau k}^{(0,1)})$ and $(q_1^{(1,0)}, ..., q_{\tau k}^{(1,0)})$ into τ vectors of k components, denoted as $(\vec{t}_1, ..., \vec{t}_\tau)$ and $(\vec{q}_1, ..., \vec{q}_\tau)$.
5. P_0 sets $\vec{c}_{0,1}^{(0)} = (\vec{g} \cdot \vec{t}_1, ..., \vec{g} \cdot \vec{t}_\tau)$;
6. P_1 sets $\vec{c}_{0,1}^{(1)} = (\vec{g} \cdot \vec{q}_1, ..., \vec{g} \cdot \vec{q}_\tau)$.

Next, each party can locally compute $\vec{c}^{(0/1)} = \vec{a}^{(0/1)} \cdot b^{(0/1)} + \sum(\vec{c}_{(0,1)}^{(0/1)} + \vec{c}_{(1,0)}^{(0/1)})$. After each party sample a random vector $\vec{r}$ over a finite field, each party will sets $a^{(0,1)} = \vec{a}^{(0/1)} \cdot \vec{r}$ and $c^{(0,1)} = \vec{c}^{(0/1)} \cdot \vec{r}$. Then parties can get a valid triple (a, b, c).

SPDZ-style online evaluation. In the online phase, the SPDZ-style protocol includes the following commands.

Input: the input command takes an input x and outputs an ASS value to each party $\langle x \rangle \leftarrow \mathsf{Input}(x)$: The parties generate an ASS sharing $\langle r \rangle \leftarrow \mathcal{F}_{\mathsf{ABB}}.\mathsf{Rand}()$, and open the value r to the party who owns the input value x. So the party will compute $\epsilon = x - r$ and broadcast ϵ. Then, all parties compute $\langle x \rangle = \langle r \rangle + \epsilon$.

Mul: On input $([\![a]\!], [\![b]\!])$ from parties, the parties will take one multiplication triple $([\![a]\!], [\![b]\!], [\![c]\!])$ and compute $[\![e]\!] = [\![x]\!] - [\![a]\!]$ and $[\![f]\!] = [\![y]\!] - [\![b]\!]$. Then, the parties compute $[\![c]\!] + f \cdot [\![a]\!] + e \cdot [\![b]\!] + e \cdot f$, which equals to $[\![x \cdot y]\!]$.

Open: on input (Open, $\langle x \rangle$) from each party, each party broadcasts $\langle x \rangle$ and recovers $x = \sum_{i \in [n]} [\![x]\!]_i$. Moreover, all parties need to run a MAC check for the opened values x in case the corrupted party opens incorrect values. The parties perform $[\![d]\!] = [\![\gamma(x)]\!] - x \cdot [\![\xi]\!]$ and check whether d equals to 0 and aborts if not. The above example is for a single evaluation. However, in implementation, the parties will perform a batch of MAC checks for better efficiency. For batch MAC check, all parties input a set of shared items $\{\langle x_1 \rangle, \ldots, \langle x_t \rangle\}$. Then, they use $\mathcal{F}_{\mathsf{ABB}}.\mathsf{Rand}()$ to sample a vector of secret-shared random values $\{[\![r_1]\!], \ldots, [\![r_t]\!]\}$, and compute $\tilde{x} = \sum_{j=1}^{t} [\![r_j]\!] \cdot [\![x_j]\!]$. Next, each party computes $[\![\sigma]\!] = \sum_{j=1}^{t} [\![r_j]\!] \cdot [\![\gamma(x_j)]\!] - \tilde{x} \cdot \xi$. Therefore, parties can perform a batched MAC check by measuring whether σ equals 0.

APPENDIX B
SECRECT SHARED SHUFFLE

The semi-honest CGP SSS protocol [12]. The CGP shuffle protocol relies on a specified correlation called *oblivious punctured matrix* (OPM). In an n-dimention OPM, a sender holds a matrix M of $n \times n$, while the receiver holds a permutation $\pi \in \mathbf{S}_n$ and a punctured matrix $\widetilde{M}$ of M, where the receiver doesn't know $M[i, \pi(i)]$ for all $i \in [n]$. From the OPM correlation, the parties can produce a correlation called *shuffle tuple*. In particular, the sender compute two n-dimension vectors $(\vec{a}, \vec{b})$ such that $\vec{a}_i = \sum_j M_{j,i}, \vec{b}_i = \sum_j M_{i,j}$ for all $i \in [n]$, and the receiver computes a n-dimension vector $\vec{\Delta}$ such that $\vec{\Delta}_i = \sum_{j \neq i} \widetilde{M}_{j,\pi(i)} - \sum_{j \neq \pi(i)} \widetilde{M}_{i,j}$.

Those *shuffle tuples* can be generated in the offline phase, and the online process of the CGP shuffling protocol is as follows. Suppose P_1 is the sender and P_0 is the receiver. Using a shuffle tuple corresponding to a permutation π, the parties can shuffle a secret-shared vector $\langle x \rangle$ as follows: P_1 sends $\vec{\delta} \leftarrow [\![\vec{x}]\!]_1 - \vec{a}$ to P_0. P_0 sets $[\![\vec{y}]\!]_0 \leftarrow \pi([\![\vec{x}]\!]_0 + \vec{\delta}) + \vec{\Delta}$ and P_1 sets $[\![\vec{y}]\!]_1 \leftarrow \vec{b}$. Clearly, $[\![\vec{y}]\!]_0 + [\![\vec{y}]\!]_1 = \pi([\![\vec{x}]\!]_0 + [\![\vec{x}]\!]_1 - \vec{a}) + \pi(\vec{a}) - \vec{b} + \vec{b} = \pi(\vec{x})$. The prior shuffling hides the underlying permutation π from P_1.

During this process, instead of directly applying the permutation π to achieve SSS, the authors split the permutation π into smaller disjoint permutations via Benes permutation network [8] to improve the performance. For n elements $x_1, ..., x_n$, the Benes network will be split into two permutations, and each permutation acts on $n/2$ elements. Every wire represents whether the element is swapped or not. Then, for each $2/n$ Benes network, it will recursively call a Benes network with half the inputs and half the outputs. The Benes network for n elements contains $2 \log n - 1$ layers, and each layer contains $n/2$ 2−element swappers.

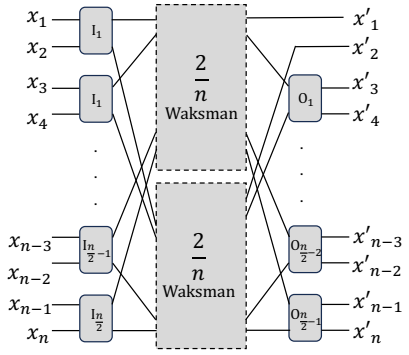

Fig. 10. Waksman Network with n inputs and outputs.

Therefore, the parties run another shuffling using another shuffle tuple (corresponding to another permutation ρ known to P_1) with roles reversed. In this manner, no party learns the underlying composed permutation.

The malicious-secure SSS protocol. Besides the semi-honest secure shuffle protocol, some malicious secure shuffle protocols [21, 29] are proposed. Those protocols design malicious CGP-style SSS protocol over ASS. Similarly, for an authenticated vector sharings $\langle\vec{x}\rangle$, the parties will perform the CGP-style shuffle protocol to get a shuffle tuple $((\pi,\vec{\delta}),(\vec{a},\vec{b}))$, where $(\pi,\vec{\delta})$ is for a receiver P_0, and $(\vec{a},\vec{b})$ is for a sender P_1. Then, parties will use their ASS sharings $\langle\vec{x}\rangle$ to do the CGP-style shuffling process as we mentioned before. P_1 sends $\vec{\delta} \leftarrow \langle\vec{x}\rangle - \vec{a}$ to P_0, The difference is that the parties can perform a MAC check to detect errors at the end of the protocol. So the checks can ensure data integrity and correct shuffling. However, those protocols are subjected to the *selective failure attack* as depicted in [49]. More concretely, a malicious sender P_1 may add errors to $\vec{\delta}$ when P_1 is expected to send $\vec{\delta} = \langle\vec{x}\rangle - \vec{a}$ to P_0. Instead of sending the correct message $\vec{\delta}$ to P_0, P_1 will change one element in $\vec{\delta}$. Then, P_1 guesses where the changed element has been permuted to. According to the post-execution check result, P_1 can learn whether the guess is correct or not. More details about *selective failure attack* are presented in Appendix. Therefore, the authors [49] propose a malicious secure SSS protocol that is also resistant to the selective failure attack. They not only propose a correlation check to defeat an incorrect correlation attack but also design a leakage-reduction mechanism to remove possible leakage to defend against the selective failure attack. Then, all the techniques used in their protocol are combined with authenticated secret sharing to formalize a malicious secure secret-shared shuffle protocol.

Concrete Optimizations. In $\Pi_{\text{m-PSI}}$, we need to use a malicious-secure SSS protocol [49] as a building block. This work also inherits the technical components of CGP protocol, and it also utilizes the Benes network [8] as a permutation structure. Based on their work, we substitute the Benes network with the Waksman network [50]. As shown in Fig. 10, the Waksman network [50] is a realization of a permutation network using exactly $n\log n - n + 1$ 2-element swappers when n is a power of 2. The Waksman network is recursive and is built by two $n/2-$input Waksman networks. Compared to the Benes network, the Waksman network achieves better trade-offs between communication and computation costs.

APPENDIX C
SECURITY PROOF

In this section, we give the ideal world definition of our proposed protocols and depict simulation-based proofs.

A. DDOPRF

Functionality $\mathcal{F}_{\text{DDOPRF}}$

Public Parameters: a prime p, a group $\mathbb{G}$ and a generator of the group g.

Private Parameter: A PRF key $k \xleftarrow{\$} \mathbb{Z}_p$ and its related ASS sharing $\langle k\rangle$; A secondary PRF key $k_s \xleftarrow{\$} \mathbb{Z}_p$ and its ASS sharing $\langle k_s\rangle$.

DDOPRF:

1. on receiving $(\text{DDOPRF}, \langle x\rangle)$ from both parties, the functionality outputs $F(k,x) = g^{\frac{1}{k+x}}$ to both parties if it does not abort. Otherwise, $\perp$ is output to both parties;
2. on receiving $(\text{DDOPRF}, \text{Dual-key}, \langle x\rangle)$ from both parties, the functionality outputs $F(k,k_s,x) = g^{\frac{k_s}{k+x}}$ to both parties if it does not abort. Otherwise, $\perp$ is output to both parties.

Fig. 11. Ideal functionality of DDOPRF.

Theorem 3. *In the $\mathcal{F}_{\text{ABB}}$, $\mathcal{F}_{\text{ABB}^+}$-hybrid model, the protocol Π_{DDOPRF} implements $\mathcal{F}_{\text{DDOPRF}}$ correctly and securely against malicious adversary.*

Proof. We can construct an ideal world simulator S_{DDOPRF} as the following:

1. S_{DDOPRF} is given the public parameter $p, \mathbb{G}, g$, and a share of x. S_{DDOPRF} simulates $\mathcal{F}_{\text{ABB}}$ and $\mathcal{F}_{\text{ABB}^+}$, chooses $k', k'_s, x' \xleftarrow{\$} \mathbb{Z}_p$ and records $\langle k'\rangle$, $\langle k'_s\rangle$ and $\langle x'\rangle$;
2. S_{DDOPRF} invokes the adversary A with $p, \mathbb{G}, g$, and $\langle x'\rangle$;
3. S_{DDOPRF} receives a $\mathcal{F}_{\text{ABB}}.\mathsf{Rand}()$ call from the adversary, generates a random number $r \in \mathbb{Z}_p$, and returns a share of r to the adversary, and S_{DOPRF} records $\langle r\rangle$;
4. S_{DDOPRF} receives the invocations to $\mathcal{F}_{\text{ABB}}.\mathsf{Mul}()$ for computing $\langle d'\rangle = \langle r\rangle \cdot (\langle k'\rangle + \langle x'\rangle)$, return a share of d' to the adversary;
5. S_{DDOPRF} receives an invocation to $\mathcal{F}_{\text{ABB}}.\mathsf{Open}$. S_{DDOPRF} does the MAC check of the inputs received in this step and the previous step, against stored shares $\langle d'\rangle, \langle k'\rangle, \langle x'\rangle, \langle r\rangle$. If all shares received from the adversary are correct, send d' to the adversary; otherwise, send abort to $\mathcal{F}_{\text{DDOPRF}}$, and abort the protocol execution with the adversary;
6. S_{DDOPRF} receives an invocation to $\mathcal{F}_{\text{ABB}}.\mathsf{Mul}()$ for computing $\langle e'\rangle \leftarrow d'^{-1} \cdot \langle r\rangle$, and S_{DDOPRF} records $\langle e'\rangle$ and returns a share of e' to the adversary; If S_{DDOPRF} receives an invocation to $\mathcal{F}_{\text{ABB}}.\mathsf{Mul}()$, it will compute $\langle e'\rangle \leftarrow \langle e'\rangle \cdot \langle k'_s\rangle$.

And S_{DDOPRF} records $\langle e' \rangle$ and returns a share to the adversary;

7. S_{DDOPRF} receives an invocation to $\mathcal{F}_{\mathsf{ABB}^+}.\mathsf{Convert}()$ for converting $\langle e' \rangle$ to $\langle\!\langle e' \rangle\!\rangle$. S_{DDOPRF} records $\langle\!\langle e' \rangle\!\rangle$ and returns a share to the adversary;
8. S_{DDOPRF} receives an invocation to $\mathcal{F}_{\mathsf{ABB}^+}.\mathsf{Open}()$ for openning $\langle\!\langle e' \rangle\!\rangle$. S_{DDOPRF} does the MAC check on all inputs received from the adversary since step 6. If the check fails, S_{DDOPRF} sends abort to $\mathcal{F}_{\mathsf{DDOPRF}}$ and abort the protocol execution with the adversary. Otherwise, S_{DDOPRF} sends its input $\langle x' \rangle$ to $\mathcal{F}_{\mathsf{DDOPRF}}$, receives the output $g^{e'}$ from $\mathcal{F}_{\mathsf{DDOPRF}}$ then passes it to the adversary.

As we can see, in this simulation: the distribution of the view of the adversary in a real execution is the same as that in the simulation because the shares are information-theoretically secure; the simulation aborts whenever an error is detected in a real execution; and the distribution of the joint output in the simulation is the same as that in a real execution. Therefore the simulation is indistinguishable from a real execution, thus $\mathcal{F}_{\mathsf{DDOPRF}}$ can be security implemented by Π_{DDOPRF}.

B. FairSec

Theorem 4. *In the $\mathcal{F}_{\mathsf{ABB}}$,$\mathcal{F}_{\mathsf{ABB}^+}$,$\mathcal{F}_{\mathsf{SSS}}$,$\mathcal{F}_{\mathsf{DDOPRF}}$-hybrid model, the protocol $\Pi_{FairSec}$ implements $\mathcal{F}_{FairSec}$ correctly and securely against malicious adversary.*

Proof. We construct an ideal world simulator S_{PSI} as the following: S_{PSI} is given its input set $\vec{x} = \{x_1, \cdots, x_n\}$. S_{PSI} simulates $\mathcal{F}_{\mathsf{ABB}}$, $\mathcal{F}_{\mathsf{ABB}^+}$, $\mathcal{F}_{\mathsf{SSS}}$, $\mathcal{F}_{\mathsf{DDOPRF}}$.

1. S_{PSI} invokes the real-world adversary A with $\vec{x}$.
2. For $i \in [1, n]$: S_{PSI} receives a $\mathcal{F}_{\mathsf{ABB}}.\mathsf{Input}(x'_i)$ call from the adversary A, then generate ASS sharing $\langle x'_i \rangle \leftarrow \mathcal{F}_{\mathsf{ABB}}.\mathsf{Input}(x'_i)$ and sends P_0's shares to A;
3. S_{PSI} sends $(\mathsf{mc\text{-}PSI}, \vec{x'})$ to the ideal functionality $\mathcal{F}_{\mathsf{mc-PSI}}$.
4. S_{PSI} selects a random $\vec{y'} = \{y'_1, \cdots, y'_n\}$, generates ASS sharings $\langle \vec{y'} \rangle$.
5. S_{PSI} receives invocations to $\mathcal{F}_{\mathsf{SSS}}$ to shuffles the shares $\langle \vec{x''} \rangle \leftarrow \mathcal{F}_{\mathsf{SSS}}(\langle \vec{x'} \rangle)$, $\langle \vec{y''} \rangle \leftarrow \mathcal{F}_{\mathsf{SSS}}(\langle \vec{y'} \rangle)$, then sends P_0's shares to A;
6. S_{PSI} receives a $\mathcal{F}_{\mathsf{ABB}^+}.\mathsf{Rand}()$ call from A, generate k, k_s and returns sharings of PRF keys $\langle k \rangle$ and $\langle k_s \rangle$ to A.
7. S_{PSI} receives an invocation to $\mathcal{F}_{\mathsf{DDOPRF}}$ to computes $Z_x = \{F(k, \vec{x''}_i)\}_{i \in [1,n]}$ and $Z_y = \{F(k, k_s, \vec{y''}_i)\}_{i \in [1,n]}$. S_{PSI} computes the multiplicative sharing of the results and returns P_0's shares to A.
8. S_{PSI} receives an invocation to $\mathcal{F}_{\mathsf{bDEC}}$ to get the shares of sequence $\langle b_0 \cdot 2^{\lambda-1} \rangle \cdots \langle b_{\lambda-1} \cdot 2^0 \rangle$, where b_i is the i-th bit of k_s (left most first) and returns P_0's shares to A.
9. S_{PSI} receives the shares $\langle b_0 \cdot 2^{\lambda-1} \rangle \cdots \langle b_{\lambda-1} \cdot 2^0 \rangle$ one by one from A and send P_1's shares to A, and they can reconstruct k_s locally;
10. If A aborts at any time in the previous steps, send abort to $\mathcal{F}_{\mathsf{m-PSI}}$, otherwise sends cardinality to $\mathcal{F}_{\mathsf{m-PSI}}$ and receives back $|\vec{x'} \cap \vec{y'}|$.
11. S_{PSI} opens Z_x and Z_y so that A can find matching elements in these two sets. S_{PSI} and A also have the sharing of $\vec{R}_{X_b}$, $\vec{R}_{Y_b}$ which corresponds to the set intersection.
12. If A aborts at any time in the previous step, send abort to $\mathcal{F}_{\mathsf{m-PSI}}$, otherwise if A invokes $\mathcal{F}_{\mathsf{f}}$, $S_P SI$ sends compute to $\mathcal{F}_{\mathsf{m-PSI}}$ and receives back $f(\vec{x'} \cap \vec{y})$, which is then forward to A.

Also, in this simulation, the adversary can only see the shares of the vectors generated during the simulation, which are information-theoretically secure. Thus, the distribution of the joint output in the simulation is the same as that in a real execution, and FairSec is secure under this situation. This situation when constructing the simulator for FairSec with corrupted P_1 is quite similar to the corrupted P_0 except that P_0 is acted by the simulator and P_1 is acted by the adversary.