Faster Than Ever: A New Lightweight Private Set Intersection and Its Variants

Guowei Ling*, Peng Tang*‡, Jinyong Shan†, Liyao Xiang*, and Weidong Qiu*‡
*School of Computer Science, Shanghai Jiao Tong University, China
Email: gw_ling@sjtu.edu.cn, tangpeng@sjtu.edu.cn, xiangliyao08@sjtu.edu.cn, qiuwd@sjtu.edu.cn
†Beijing Smartchip Microelectronics Technology Co., Ltd., China
Email: shanjinyong@sgchip.sgcc.com.cn
‡Corresponding authors.

***Abstract*—In this work, we present a new lightweight two-party Private Set Intersection (PSI) paradigm in both the semi-honest and malicious models. It requires only a small number of base Oblivious Transfers (OTs), along with a single Oblivious Key-Value Store (OKVS) encoding and a number of decodings equal to the sender input size. All computations (except for the base OTs) can be implemented using efficient hash and bitwise operations. Furthermore, we extend the proposed PSI protocol to circuit PSI and, subsequently, to several PSI variants, including PSI-cardinality, PSI-sum, and Private Join and Compute (PJC). All proposed protocols are evaluated under both LAN and WAN settings, with performance compared against existing works. Experimental results demonstrate that the proposed PSI achieves about $1.5\times$ faster runtime than the most efficient Vector Oblivious Linear Evaluation (VOLE)-based PSI, while maintaining consistently lower communication overhead under identical settings. For circuit PSI, it is up to $3.6\times$ faster and reduces communication by a factor of 1.5 compared to VOLE-based circuit PSI constructions. In the cases of PSI-cardinality and PSI-sum, they achieve speedups of up to $12.2\times$ and $10\times$, respectively, while incurring only moderate communication overhead. For PJC, the proposed protocol outperforms prior work by $731\times$ in runtime and achieves a $3.2\times$ reduction in communication, maintaining high efficiency even under a low-bandwidth condition. The performance under the unbalanced setting is also evaluated, showing that our PSI achieves at most two orders of magnitude improvement in the LAN setting compared with existing unbalanced PSI protocols, and gradually gains superiority under the WAN setting as the receiver set size increases.**

I. Introduction

In recent years, there has been a growing interest in enabling independent parties to compute insights over their combined data collaboratively. In many scenarios, the data held by each party is sensitive and should remain confidential. As a result, the objective is to ensure that each party learns only the intended outputs and nothing beyond. This challenge has garnered attention from several major organizations, including Google [1], Meta [2], Signal [3], and Apple [4], all of which have actively contributed to research and development in this area. In the context of privacy-preserving collaborative computation among multiple parties, one particularly important task is Private Set Intersection (PSI), which has garnered considerable attention in recent years due to its wide range of practical applications [5]–[9]. In a PSI protocol, two parties, the sender and the receiver, each hold private sets X and Y, respectively. The goal is to securely compute the intersection $X \cap Y$ without revealing any additional information beyond the sizes of the input sets.

Modern PSI protocols are primarily constructed using two mainstream approaches: Oblivious Transfer Extension (OTE)-based PSI [10]–[14] and Vector Oblivious Linear Evaluation (VOLE)-based PSI [15]–[19]. Over the years, VOLE-based PSI has generally come to be regarded as more efficient in terms of performance. Its main construction, originating from [15], typically relies on a data structure known as Oblivious Key-Value Stores (OKVS) [20]. This type of PSI protocol is highly efficient, typically completing the intersection computation for inputs of size 2^{20} within approximately one second and requiring only tens of megabytes of communication. As a result, PSI protocol development has increasingly converged on the VOLE-based paradigm, with progress slowing in the past few years. In fact, since the introduction of this framework by Rindal and Schoppmann [15], subsequent works [16]–[19] have largely followed the same paradigm, differing only in the choice of underlying VOLE protocols or the specific OKVS constructions employed. For example, the protocol presented in [16], which is widely regarded as the state-of-the-art PSI protocol, is built upon [15] with improved constructions of VOLE and OKVS. Proposing a new PSI paradigm, rather than continuing along the VOLE-based PSI direction, appears particularly challenging, given that existing VOLE-based PSI protocols are already highly efficient and exhibit low communication overhead. In this work, we take on the challenge of departing from the VOLE-based paradigm of [15], and present a novel PSI construction that achieves improved performance. We demonstrate that, under comparable settings, our PSI protocol consistently outperforms the state-of-the-art OTE-based [12] and VOLE-based [15]–[17] PSI protocols in both LAN and WAN environments. Moreover, our paradigm incurs strictly less communication than the VOLE-

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.240131
www.ndss-symposium.org

based PSI paradigm.

In some specific scenarios, the objective of the sender and the receiver may not be to learn the intersection itself, but rather to compute specific functions over the intersection. For example, in the context of evaluating advertising effectiveness by correlating online ad impressions with offline credit card transactions [5], the advertiser is primarily interested in the total sales linked to a particular set of credit cards used at designated vendors, rather than in the details of individual transactions. A general approach is circuit PSI [22]–[25], where the output consists of secret shares of the inputs and their corresponding payloads from both parties, rather than the intersection itself. In theory, these secret shares enable the execution of arbitrary downstream computations via secure Multi-Party Computation (MPC). To address these practical scenarios, we extend our PSI protocol to a circuit PSI protocol, which is then used to implement several important PSI variants, including PSI-cardinality, PSI-sum, and Private Join and Compute (PJC). Furthermore, a comparative evaluation of the proposed circuit PSI protocol and these PSI variants against existing works is conducted to demonstrate the advantages of the construction.

A. Our Contributions

We summarize the contributions of this work as follows:

- It proposes a new two-party PSI paradigm that utilizes Oblivious Transfer (OT) and OKVS, and supports both the semi-honest and malicious models. Any improvements to OT or OKVS would directly benefit the performance of our protocol. Interestingly, unlike prior works [10]–[19], [26], our protocol does not require expanding κ instances of OT into a large number of pseudorandom correlations, such as OTE, VOLE, or Beaver triples. Although this extension is already highly communication-efficient due to the use of silent techniques [27]–[30], our protocol still achieves strictly lower communication than VOLE-based protocols [15]–[17] under identical settings, while naturally avoiding their associated computational overhead. For example, according to our evaluation, completing a silent VOLE protocol [29] with an input size of 2^{24} takes approximately 6 seconds, whereas our protocol avoids this cost entirely.
- The proposed PSI can be extended to circuit PSI, enabling the realization of various PSI variants. A comprehensive comparison with existing works is conducted in terms of runtime under both LAN and WAN settings, as well as communication. The results are summarized in Table I.
  - **PSI**. The proposed protocol achieves approximately $1.5\times$ faster performance compared to the VOLE-based PSI [15]–[17]. The advantage diminishes as bandwidth decreases, since our communication, while smaller than that of VOLE-based PSI, is not significantly so. Nevertheless, our PSI incurs strictly less communication under identical settings. Moreover, compared to the state-of-the-art OTE-based PSI protocol [12], our protocol achieves more than $3\times$ faster runtime and reduces communication by a factor of 4.
  - **Circuit PSI**. The circuit PSI proposed protocol is $3.6\times$ faster than the VOLE-based circuit PSI [15]–[17] and reduces the communication cost by a factor of 1.5. Similar to the standard PSI setting, as the bandwidth decreases, the performance of the protocol becomes increasingly constrained by the network. Nevertheless, our protocol maintains strong performance even in low-bandwidth environments. Even at 10 Mbps, the proposed protocol remains $1.7\times$ faster. We further compare our circuit PSI with several recent circuit PSI constructions [22]–[24] to demonstrate its competitive performance. For example, compared to PSTY [22], our protocol achieves more than an $8\times$ speedup and reduces computational overhead by over $30\times$.
  - **PSI-cardinality and PSI-sum**. The proposed protocols outperform the state-of-the-art protocol [21] by $12.2\times$ and $10\times$ in PSI-cardinality and PSI-sum, respectively. A drawback is that the protocols incur approximately 50% more communication overhead than [21]. Nevertheless, due to its significant computational efficiency advantage, our protocol is slower than [21] only under very low-bandwidth conditions. Specifically, at 10 Mbps, its performance becomes slightly inferior to [21], but it remains faster under bandwidth settings of 100 Mbps or higher. Furthermore, our protocol incurs lower computation and communication overhead than other advanced baselines [22], [24], [31].
  - **PJC**. The proposed PJC protocol is $731\times$ faster and achieves a $3.2\times$ reduction in communication compared to the existing PJC protocol [5]. Even under a 10 Mbps setting, it still achieves an $18\times$ performance improvement due to its computational efficiency.
- In addition to the functional variants, we also evaluate our PSI under the unbalanced setting and compare it with the state-of-the-art unbalanced PSI protocols [32], [33]. Our PSI demonstrates a clear advantage in the LAN setting, achieving up to one or two orders of magnitude speedup over [32], [33]. As the receiver set size increases, it also gains superiority under WAN conditions. For example, when the receiver input exceeds 2^{14}, our PSI outperforms [32], [33] in the 100 Mbps setting. In addition, we observe that when the size of the sender set is less than about 2^6 (i.e., 64) times that of the receiver, our PSI remains the fastest even under the 10 Mbps setting.
- All implementations developed in this work are publicly available for reproducibility at https://github.com/ShallMate/OurPSI.

B. Notation

All notations used throughout this paper adhere to the definitions provided below, unless otherwise stated. Let κ denote the computational security parameter and λ the statistical security parameter. The sender is denoted by $\mathcal{S}$ and the receiver by $\mathcal{R}$. The input set of $\mathcal{S}$ is denoted by X, and that of $\mathcal{R}$ by

TABLE I: Comparison between our work and state-of-the-art (SOTA) works across various aspects. If the existing work is optimal, it is highlighted in green; if the proposed work is optimal, it is highlighted in red.

Functionality	SOTA Baseline	Execution Time Comparison			Communication Comparison	Detailed Results
		LAN	100Mbps	10Mbps		
PSI	VOLE-based PSI [15]–[17]	$\approx 1.5\times \uparrow$	$\approx 1.2\times \uparrow$	$\approx 1.03\times \uparrow$	$\approx 1.02\times \uparrow$	Table III
Circuit PSI	VOLE-based CPSI [15]–[17]	$\approx 3.6\times \uparrow$	$\approx 2.4\times \uparrow$	$\approx 1.7\times \uparrow$	$\approx 1.5\times \uparrow$	Table IV
PSI-cardinality	CZZ$^+$ [21]	$\approx 12.2\times \uparrow$	$\approx 3.6\times \uparrow$	$\approx 0.12\times \downarrow$	$\approx 0.49\times \downarrow$	Table V
PSI-sum	CZZ$^+$ [21]	$\approx 10\times \uparrow$	$\approx 3\times \uparrow$	$\approx 0.1\times \downarrow$	$\approx 0.48\times \downarrow$	Table VI
PJC	IKN$^+$ [5]	$\approx 731\times \uparrow$	$\approx 121\times \uparrow$	$\approx 18\times \uparrow$	$\approx 3.2\times \uparrow$	Table VII

Y. The corresponding payloads are denoted by $\tilde{X}$ and $\tilde{Y}$, respectively. Let the input sizes of $\mathcal{S}$ and $\mathcal{R}$ be denoted by n_s and n_r, respectively. For simplicity of presentation, we set $n = n_s = n_r$ throughout this paper, except when evaluating the PSI performance under the unbalanced setting. ϵ denotes the redundancy rate of the OKVS. The notation $[a]$ denotes the set $\{1, 2, \ldots, a\}$. The symbol $:=$ is used to denote assignment, while $=$ indicates equality. $||$ denotes bitwise concatenation. Let out denote the output length (i.e., the mask length) of the PSI protocol, and let ϵ denote the redundancy rate of the OKVS. If a capital letter such as A denotes a set, then the corresponding lowercase a_i refers to its i-th element. Similarly, if s is a binary string, then $s[i]$ denotes the value of its i-th bit. For a hash table T, we use $T[i]$ to denote the set of all elements stored at position i.

II. Technical Overview

We begin by separately reviewing the most efficient OTE-based and VOLE-based PSI protocols, which collectively motivate the design of our PSI. The protocols reviewed in this section, as well as our PSI construction, are presented in a simplified and abstracted form for exposition, rather than in their formal descriptions.

OTE-based PSI. To the best of our knowledge, KKRT [12] remains the most efficient OTE-based PSI protocol and was long regarded as the state-of-the-art prior to the advent of VOLE-based PSI [15]–[17]. The overall workflow of the protocol is abstracted and presented in Figure 1.

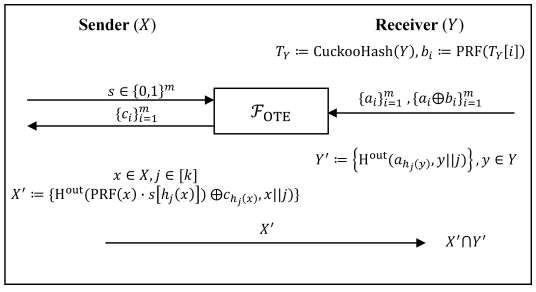

Fig. 1: High-level illustration of OTE-based PSI [12].

First, $\mathcal{R}$ inserts its set Y into a cuckoo hash table T_Y of length m using hash functions $\{h_j\}_{j=1}^k$. For each $T_Y[i]$, $\mathcal{R}$ computes a pseudorandom function value $b_i := \mathsf{PRF}(T_Y[i])$. Next, the parties perform OTE with swapped roles: $\mathcal{S}$ samples $s \in \{0,1\}^m$, $\mathcal{R}$ inputs $\{a_i, a_i \oplus b_i\}_{i=1}^m$, and $\mathcal{S}$ receives $\{c_i\}_{i=1}^m$. Finally, $\mathcal{S}$ computes and sends $X' := \{\mathsf{H}^{\text{out}}((\mathsf{PRF}(x) \cdot s[h_j(x)]) \oplus c_{h_j(x)}, x||j)\}$, while $\mathcal{R}$ sets $Y' := \{\mathsf{H}^{\text{out}}(a_{h_j(y)}, y||j)\}$, where H^{out} denotes a random oracle. Note that $\mathcal{R}$ computes each mask only once by recording the hash index j used during the insertion into the cuckoo hash table. $\mathcal{R}$ computes $X' \cap Y'$ to reveal the intersection result. For each $x \in X \cap Y$, let $y = x$ be the element inserted into the cuckoo hash table using h_j. Then, we have $(\mathsf{PRF}(x) \cdot s[h_j(x)]) \oplus c_{h_j(x)} = a_{h_j(y)}$. This equality holds for both $s[h_j(x)] = 0$ and $s[h_j(x)] = 1$. This is because when $s[h_j(x)] = 0$, the left-hand side reduces to $c_{h_j(x)}$, which equals $a_{h_j(y)}$ due to the correctness of OT. When $s[h_j(x)] = 1$, the left-hand side becomes $\mathsf{PRF}(x) \oplus (a_{h_j(x)} \oplus \mathsf{PRF}(x)) = a_{h_j(x)} = a_{h_j(y)}$, so the equality still holds.

VOLE-based PSI. We now turn to reviewing the VOLE-based PSI paradigm [15], [16]. An overview of the protocol workflow is depicted in Figure 2.

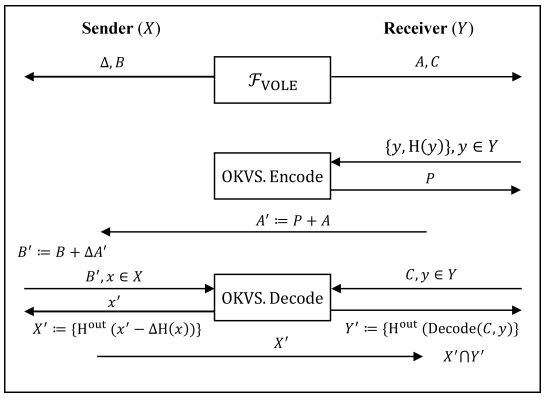

Fig. 2: High-level illustration of VOLE-based PSI [15], [16].

First, $\mathcal{S}$ and $\mathcal{R}$ execute a VOLE protocol [29], [34] and obtain Δ, B and A, C, respectively, where $C = \Delta A + B$. Then, $\mathcal{R}$ executes the OKVS encoding algorithm to encode the key-value pairs $\{y, \mathsf{H}(y)\}$ into P, computes $A' := P + A$, and sends A' to $\mathcal{S}$. Subsequently, $\mathcal{S}$ computes $X' := \{\mathsf{H}^{\text{out}}(\mathsf{Decode}(B', x) - \Delta \mathsf{H}(x))\}$ for each $x \in X$ and sends X' to $\mathcal{R}$, where $B' = B + \Delta A'$. Finally, $\mathcal{R}$ computes $Y' := \{\mathsf{H}^{\text{out}}(\mathsf{Decode}(C, y)) \mid y \in Y\}$ and determines the intersection by computing $X' \cap Y'$. When $x = y \in X \cap Y$, we have

$$\begin{aligned}\mathsf{Decode}(B', x) - \Delta\mathsf{H}(x) &= \mathsf{Decode}(B + \Delta(P + A), x) - \Delta\mathsf{H}(x)\\ &= \mathsf{Decode}(C + \Delta P, x) - \Delta\mathsf{H}(x)\\ &= \mathsf{Decode}(C, x) = \mathsf{Decode}(C, y)\end{aligned}$$

Interestingly, VOLE-based and OTE-based PSI protocols share fundamentally similar cores, despite exhibiting entirely

different presentations. At a high level, OKVS plays the role of cuckoo hashing as in the OTE-based PSI protocol [12], while VOLE replaces the role of OTE. The term $P+A$ serves a role analogous to $\{a_i, a_i \oplus b_i\}$, ensuring that any common element between the two parties deterministically leads to the same masked value. Therefore, after analyzing the essential similarity between the two protocols, we can further integrate the strengths of each to design an improved protocol.

Our PSI. We retain OKVS from the VOLE-based PSI to align elements, avoiding the k-fold communication overhead in KKRT caused by using k hash functions. Furthermore, we partially incorporate the masking strategy of KKRT, which allows the majority of our protocol's computations to be efficiently executed using highly optimized hash functions, such as SHA256 via Intel SHA2-NI intrinsics. The high-level workflow of our PSI is illustrated in Figure 3.

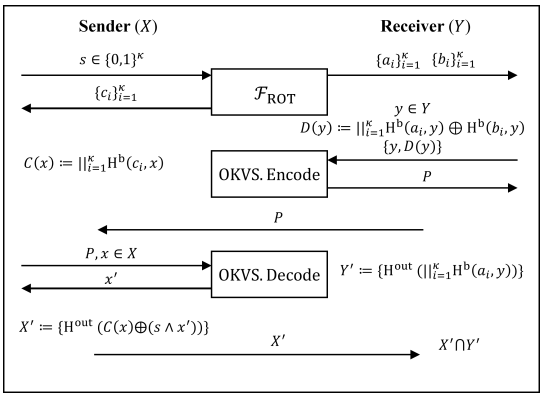

Fig. 3: A simplified version of our PSI protocol.

As a first step, the parties perform κ base random OTs with swapped roles: $\mathcal{R}$ obtains $\{(a_i, b_i)\}_{i=1}^\kappa$, and $\mathcal{S}$, with randomly chosen choice bits $s \in \{0,1\}^\kappa$, receives the corresponding messages $\{c_i\}_{i=1}^\kappa$. Then, for each $y \in Y$, $\mathcal{R}$ computes $D(y) := ||_{i=1}^\kappa \mathsf{H}^{\mathsf{b}}(a_i, y) \oplus \mathsf{H}^{\mathsf{b}}(b_i, y)$, where H^{b} denotes a random oracle mapping to $\{0,1\}$. $\mathcal{S}$ computes $C(x) := ||_{i=1}^\kappa \mathsf{H}^{\mathsf{b}}(c_i, x)$ for each $x \in X$. Since $\mathcal{S}$ obtains only one of a_i or b_i for each i, every bit of $D(y)$ appears uniformly random from $\mathcal{S}$. This is straightforward to understand: H^{b} is modeled as an oracle, while in practice it is instantiated using SHA-256 by taking the least significant bit of the output. Consequently, each bit of $D(y)$ is computed as the XOR of a deterministic bit and a pseudorandom bit for $\mathcal{S}$, rendering it indistinguishable from uniform over $\mathbb{F}_2$. Therefore, $\mathcal{R}$ can directly encode the set $\{(y, D(y))\}$ into the OKVS P. Given that $D(y)$ is indistinguishable from random for $\mathcal{S}$, it cannot reverse-engineer the encoding, owing to the properties of the OKVS. Finally, $\mathcal{S}$ computes and sends $X' := \{\mathsf{H}^{\mathsf{out}}(C(x) \oplus (s \wedge \mathsf{Decode}(P, x)))\}$, and $\mathcal{R}$ sets $Y' := \{\mathsf{H}^{\mathsf{out}}(\|_{i=1}^\kappa \mathsf{H}^{\mathsf{b}}(a_i, y))\}$. $\mathcal{R}$ can obtain the intersection by computing $X' \cap Y'$. For $x = y \in X \cap Y$, it is necessary to verify that $C(x) \oplus (s \wedge \mathsf{Decode}(P, x)) = ||_{i=1}^\kappa \mathsf{H}^{\mathsf{b}}(a_i, y)$.

For simplicity of analysis, we consider a single bit position. We have:

$$\mathsf{H}^{\mathsf{b}}(c_i, x) \oplus \big(s[i] \wedge \big(\mathsf{H}^{\mathsf{b}}(a_i, x) \oplus \mathsf{H}^{\mathsf{b}}(b_i, x)\big)\big) = \mathsf{H}^{\mathsf{b}}(a_i, y).$$

It is easy to verify that this equality holds regardless of whether $s[i] = 0$ or $s[i] = 1$. The protocol involves only base OTs, a single OKVS encoding and a number of decodings equal to the sender input size, with all other computations efficiently realized via highly optimized hash instruction set extensions, resulting in high overall efficiency.

Differences from previous PSI paradigms. Our PSI represents a new two-party PSI paradigm. It no longer requires OTE or VOLE, but instead relies only on at most κ base random OTs. Previous efficient PSI protocols typically relied on a large number of pseudorandom correlations, such as OTE, VOLE, OLEs, and Beaver triples. In contrast, our construction eliminates the need, fundamentally distinguishing it from all existing efficient PSI designs. As mentioned earlier, while silent techniques [27]–[30] can greatly reduce communication, their computational cost remains non-negligible. Its underlying components are reduced to base random OTs and the OKVS, rather than following the VOLE-based PSI paradigm adopted in previous works [15]–[19]. To the best of our knowledge, our PSI paradigm is the first to avoid all of these pseudorandom correlations, requiring only base random OTs (at most 128 random OTs), while still achieving performance that surpasses prior works.

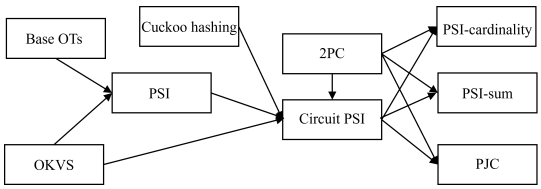

Fig. 4: Our modular technical roadmap for PSI and its variants.

Our circuit PSI. We demonstrate how our PSI can be extended to a circuit PSI, and subsequently to various PSI variants. The technical roadmap is illustrated in Figure 4. To extend the PSI to circuit PSI, we additionally employ OKVS in combination with cuckoo hashing, along with a generic two-party computation functionality, as illustrated in Figure 5. This extension was also employed in [15]–[17]. Although this transformation is not our original contribution, we demonstrate that our PSI also supports this extensibility property, even though it departs from the VOLE-based PSI paradigm.

Parameters: The input size and output size of $\mathcal{S}$ are denoted by in_s and out_s, respectively, and those of $\mathcal{R}$ are denoted by in_r and out_r. The functionality is defined by a circuit C that takes inputs from both parties and produces outputs for each, i.e., $C : \{0,1\}^{\mathsf{in}_s + \mathsf{in}_r} \to \{0,1\}^{\mathsf{out}_s + \mathsf{out}_r}$.
Inputs: $\mathcal{S}$ provides a private input $X \in \{0,1\}^{\mathsf{in}_s}$, and $\mathcal{R}$ provides a private input $Y \in \{0,1\}^{\mathsf{in}_r}$.
Output: Compute $(Z_1, Z_2) := C(X, Y)$, return Z_1 to $\mathcal{S}$ and Z_2 to $\mathcal{R}$.

Fig. 5: Ideal functionality for $\mathcal{F}_{\mathsf{2PC}}$ for two party computation.

Before describing our circuit PSI, we present a concept related to PSI: the PSI mask. In simple terms, $\mathcal{S}$ holds a set X and $\mathcal{R}$ holds a set Y. Let $\mathsf{F} : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^{\mathsf{out}}$. Their intersection process can be described as follows: $\mathcal{S}$ computes $X' = \{\mathsf{F}(k,x) \mid x \in X\}$ and sends it to $\mathcal{R}$, while $\mathcal{R}$ evaluates $Y' = \{\mathsf{F}(k,y) \mid y \in Y\}$ without knowing $k \leftarrow \{0,1\}^\kappa$. We refer to X' and Y' as the masks of X and Y, respectively. A high-level overview of our circuit PSI is illustrated in Figure 6. In circuit PSI, in addition to their respective input sets X and Y, both parties also provide the corresponding payload sets $\tilde{X}$ and $\tilde{Y}$ as input. First, $\mathcal{R}$ constructs a cuckoo hash table T_Y of size m by setting $\mathsf{H}(y\|j,r) := T_Y[h_j(y)]$. $\mathcal{S}$ constructs a simple hash table T_X such that $\mathsf{H}(x\|j,r) \in T_X[h_j(x)]$. Next, both parties run our PSI over T_X and T_Y, and respectively obtain the masked sets T'_X and T'_Y. However, $\mathcal{S}$ does not send the masks T'_X to $\mathcal{R}$. Then, $\mathcal{S}$ randomly samples $\{r_i\}_{i=1}^m$ and $\{\tau_i\}_{i=1}^m$. For all $t_x \in T_X$, $\mathcal{S}$ encodes the pair $\{(t_x, t'_x || t_{\tilde{x}} \oplus \tau_i)\}$ into the OKVS P and sends it to $\mathcal{R}$, where t'_x and $t_{\tilde{x}}$ denote the PSI mask and the corresponding payload associated with t_x, respectively. Subsequently, for each $t_y \in T_Y$, $\mathcal{R}$ can compute $r'_i \| t'_i := \mathsf{Decode}(P, t_y) \oplus t'_y$. Finally, for each $i \in [m]$, both parties invoke $\mathcal{F}_{\mathsf{2PC}}$ to obtain secret shares (q_i^0, q_i^1) of q_i, where $q_i := 1$ if $r_i = r'_i$, and $q_i := 0$ otherwise. They also compute shares (z_i^0, z_i^1) of $z_i := q_i \cdot ((\tau'_i \oplus \tau_i) \| t_{\tilde{y}})$. The shares of q_i and z_i constitute the output of the circuit PSI.

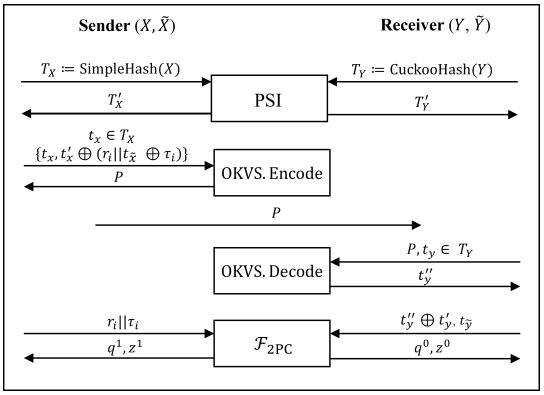

Fig. 6: A simplified version of our circuit PSI protocol.

In contrast to [15]–[17], which did not further develop support for various PSI variants, our work presents dedicated circuits for multiple PSI variants, including PSI-cardinality, PSI-sum, and PJC, and provides implementations and thorough performance evaluations for each.

III. Related Work

This section provides a review of the evolution of the PSI protocol and its variants.

PSI. The term PSI was coined by Freedman et al. [35]. However, PSI protocols predate this terminology and can be traced back to earlier constructions based on Diffie-Hellman (DH) key agreement [36], [37]. This class of protocols was initially considered inefficient due to their exclusive reliance on public-key operations. However, with continuous optimizations in elliptic curves [38], [39], the efficiency of DH-based PSI has been significantly improved. Modern DH-based PSI protocols [21], [40] can complete the intersection of 2^{20} elements within tens of seconds. Pinkas et al. [10] constructed an efficient PSI protocol based on OTE [41], which is considered the origin of the highly efficient class of OTE-based PSI protocols. Since then, a series of efficient PSI protocols [11]–[14] have been proposed, with [12] being the most efficient among these. However, compared with DH-based PSI, these protocols trade communication overhead for significantly higher computational efficiency. This situation has been substantially improved with the introduction of OKVS [20], which offers a convenient representation of private sets to facilitate efficient set intersection. Notably, OKVS was initially introduced to address the challenge that cuckoo hashing [42] posed for achieving malicious security. For further details, see [20]. As a result, early OKVS-based PSI protocols incurred considerable communication overhead. This issue was quickly addressed by Garimella et al. [43], who optimized the underlying design. Building on this improvement, Rindal et al. [15] combined OKVS with a VOLE protocol [44] to construct a PSI protocol that supports both the semi-honest and malicious security models. This architecture has established the standard VOLE-based PSI paradigm, with follow-up works [16], [17] primarily enhancing the underlying VOLE or OKVS components while preserving the overall design framework. In general, VOLE-based PSI protocols [15]–[17] are collectively regarded as the state-of-the-art in the context of PSI.

Circuit PSI. Circuit PSI is a variant of PSI where the goal is to compute a function of the intersection. The circuit PSI proposed by Huang et al. [45] performs several comparisons proportional to the input size times its logarithm, and its overall circuit size scales accordingly. In contrast, the circuit PSI by Pinkas et al. [11] employs a different strategy: one party applies cuckoo hashing to distribute its elements across a linear number of bins, while the other party uses simple hashing that maps only a logarithmic number of elements into each bin. This design yields a circuit whose size scales with the input size times the logarithm of the input size, divided by the double logarithm of the input size. Ciampi et al. [46] follow a different paradigm by leveraging OTE to privately test membership in each bin. The results of these tests are then fed into a comparison circuit that performs a linear number of comparisons. While the circuit itself is compact, the total communication complexity remains in the same asymptotic order as that of [11]. Soon after, Pinkas et al. [22] were the first to demonstrate how to construct a circuit PSI with linear communication complexity. However, this protocol suffers from super-linear computational complexity. Chandran et al. [23] quickly addressed this issue and proposed a circuit PSI that achieves both linear communication and linear computational complexity. Subsequently, Rindal et al. [15] extended the VOLE-based PSI to the setting of circuit PSI, which remains the state-of-the-art construction for circuit PSI. Mahdavi et al. [24] introduced a homomorphic-encryption-based circuit PSI that is particularly efficient in

highly unbalanced input scenarios. Recently, Yang et al. [25] proposed the first maliciously secure circuit-based PSI, but its performance remains far behind semi-honest protocols.

PSI-cardinality. PSI-cardinality reveals only the size of the intersection rather than the intersection itself. Several early constructions [47]–[49] achieve this functionality using public-key operations while ensuring linear complexity. Alternatively, this functionality can also be realized through the use of circuit PSI protocols such as [22], [24], [31]. Recently, Chen et al. [21] introduced a suite of private set operations, including PSI-cardinality. Their construction achieves the lowest known communication overhead among existing PSI-cardinality protocols, making it particularly well-suited for deployment in environments with very low bandwidth.

PSI-sum. PSI-sum computes the aggregate sum of the payload values corresponding to all elements held by the sender that appear in the intersection with the receiver's set. A representative example is the protocol proposed by Ion et al. [5], which aims to compute the total amount of purchases made by users who were exposed to specific advertisements. However, the performance of the protocol is limited, as it heavily relies on computationally expensive public-key operations. Similar to PSI-cardinality, this functionality can also be realized using circuit PSI protocols, such as [22], [24]. In addition, [21] also achieves this functionality with low communication overhead and demonstrates advantages in low-bandwidth environments.

PJC. PJC is a more general form of PSI-sum, which computes the sum of the products of the payloads associated with the elements in the intersection held by both parties. This idea was first introduced in [5], [50]. Subsequently, Lepoint et al. [51] formally defined this class of protocols under the term PJC. Naturally, PJC can also be realized using circuit PSI protocols, as demonstrated in [24], [52].

Unbalanced PSI. Unbalanced PSI is a special case of PSI where the set held by one party is significantly smaller than the set held by the other. The current unbalanced PSI protocols are primarily constructed based on fully homomorphic encryption. Chen et al. [53] introduced optimizations to reduce the multiplicative depth of the function evaluated homomorphically, thereby enhancing efficiency. Furthermore, Chen et al. [54], and Cong et al. [32] employ a combination of OPRF and fully homomorphic encryption, building upon [53]. Mahdavi et al. [24] adopted a constant-weight encoding method, which significantly improves performance in the offline phase, but communication and computation in the online phase are slower than those of [32]. Recently, Chielle et al. [33] proposed an unbalanced PSI protocol specifically optimized for repeated executions, which achieves notably high efficiency when the receiver input set is very small. Overall, [32], [33] represent the state-of-the-art unbalanced PSI protocols.

IV. Preliminaries

A. Oblivious Transfer (OT)

Originally introduced by Rabin [55], OT is a foundational primitive in the field of MPC. In the standard 1-out-of-2 OT setting, the sender $\mathcal{S}$ holds two input messages (m_0, m_1), while the receiver $\mathcal{R}$ possesses a selection bit $b \in \{0,1\}$. At the conclusion of the protocol, $\mathcal{R}$ obtains m_b without learning any information about m_{1-b}, and $\mathcal{S}$ remains unaware of the $\mathcal{R}$'s choice bit b.

Parameters: Message length L.
Input: A random choice bit $b \in \{0,1\}$ for $\mathcal{R}$.
Output: The functionality samples two random messages $m_0, m_1 \in \{0,1\}^L$. $\mathcal{R}$ obtains m_b, and $\mathcal{S}$ obtains (m_0, m_1).

Fig. 7: Ideal functionality $\mathcal{F}_{\mathsf{ROT}}$ for random 1-out-of-2 OT.

Ishai et al. [41] proposed the OTE technique, which enables the execution of a large number of OT instances using only κ base OTs, with the remaining operations relying solely on efficient symmetric-key primitives. In this work, we show that OT extension is no longer necessary, and that our PSI can be efficiently realized using only the base random OT instances. The ideal functionality $\mathcal{F}_{\mathsf{ROT}}$ for random OT is specified in Figure 7. We denote by $\mathcal{F}_{\mathsf{ROT}}^t$ the functionality that makes t independent invocations of $\mathcal{F}_{\mathsf{ROT}}$.

B. Private Set Intersection (PSI)

The PSI functionality considered in this work follows the definition used in VOLE-based PSI [15], as illustrated in Figure 8. This is a widely adopted definition of PSI, although it allows $\mathcal{R}$ to evaluate on n_r' elements, which may be slightly larger than n_r. A detailed discussion on this aspect can be found in the recent work by Han et al. [18].

Parameters: The input set size of $\mathcal{S}$ is n_s, and that of $\mathcal{R}$ is n_r. Let $n_r' \geq n_r$ be a public parameter.
Inputs: $\mathcal{S}$ provides a set X. $\mathcal{R}$ provides a set Y. If $|X| > n_s$, abort. If $\mathcal{R}$ is malicious and $|Y| > n_r'$, then abort. If $\mathcal{R}$ is honest and $|Y| > n_r$, then abort.
Output: $\mathcal{R}$ receives the set intersection $I := X \cap Y$.

Fig. 8: Ideal functionality for $\mathcal{F}_{\mathsf{PSI}}$ for two-party PSI.

In many scenarios, the two parties are not interested in computing the intersection itself, but rather in using it to perform some downstream task. A general approach is to employ a circuit PSI protocol, allowing both parties to obtain, for each element, secret shares of both its intersection status and its associated payload. The ideal functionality of the circuit PSI is depicted in Figure 9. Beyond circuit PSI, there

Parameters: The input set size of $\mathcal{S}$ is n_s, and that of $\mathcal{R}$ is n_r. A function $\mathsf{R} : \{\{0,1\}^*\}^{n_r} \to (\pi : [n_r] \to [m])$ that outputs an injective function π.
Inputs: $\mathcal{S}$ provides $(X, \tilde{X})$. $\mathcal{R}$ provides $(Y, \tilde{Y})$.
Outputs: The functionality computes $\pi \leftarrow \mathsf{R}(Y)$ and uniformly samples $Q^0, Q^1 \in \{0,1\}^m$ and $Z^0, Z^1 \in \left\{\{0,1\}^{|\tilde{x}|+|\tilde{y}|}\right\}^m$, such that for each $i' = \pi(i)$, it holds that $q_{i'}^0 \oplus q_{i'}^1 = 1$ and $z_{i'}^0 \oplus z_{i'}^1 = (\tilde{x}_i \| \tilde{y}_i)$ if there exists $y_i \in Y$ and $x_j \in X$ such that $y_i = x_j$; otherwise, $q_{i'}^0 \oplus q_{i'}^1 = 0$ and $z_{i'}^0 \oplus z_{i'}^1 = 0$. The functionality outputs (Q^0, Z^0, π) to $\mathcal{R}$ and (Q^1, Z^1) to $\mathcal{S}$.

Fig. 9: Ideal functionality for $\mathcal{F}_{\mathsf{CPSI}}$ for circuit PSI.

are also several concrete PSI variants designed for particular use cases. For example, PSI-cardinality is a specific PSI variant that reveals only the size of the intersection, as shown in Figure 10.

Parameters: The input set size of $\mathcal{S}$ is n_s, and that of $\mathcal{R}$ is n_r.
Inputs: $\mathcal{S}$ provides a set X. $\mathcal{R}$ provides a set Y.
Output: $\mathcal{R}$ receives the set intersection $I := |X \cap Y|$.

Fig. 10: Ideal functionality for $\mathcal{F}_{\mathsf{PSICA}}$ for PSI-cardinality.

In addition, we also realize PSI-sum and PJC, which compute the sum of $\mathcal{S}$'s payloads over the intersection and the inner product of the payloads from both parties over the intersection, respectively. The ideal functionality of PSI-sum is defined in Figure 11, while that of PJC is presented in Figure 12. In PSI-sum, only $\mathcal{R}$'s input set is associated with payloads, whereas in PJC, both parties have associated payloads.

Parameters: The input set size of $\mathcal{S}$ is n_s, and that of $\mathcal{R}$ is n_r.
Inputs: $\mathcal{S}$ provides $(X, \tilde{X})$. $\mathcal{R}$ provides Y.
Outputs: $\mathcal{R}$ receives the result $\mathsf{sum} := \sum_{x \in X \cap Y} \tilde{x}$.

Fig. 11: Ideal functionality for $\mathcal{F}_{\mathsf{PSISum}}$ for PSI-sum.

Parameters: The input set size of $\mathcal{S}$ is n_s, and that of $\mathcal{R}$ is n_r.
Inputs: $\mathcal{S}$ provides $(X, \tilde{X})$. $\mathcal{R}$ provides $(Y, \tilde{Y})$.
Outputs: $\mathcal{R}$ receives the result $\mathsf{sum} := \sum_{x_i = y_j} \tilde{x}_i \tilde{y}_j$.

Fig. 12: Ideal functionality for $\mathcal{F}_{\mathsf{PJC}}$ for PJC.

PSI-cardinality, PSI-sum, and PJC can be realized either through circuit PSI or dedicated protocol designs. This work adopts the circuit PSI approach to implement these functionalities.

C. Oblivious Key-Value Stores (OKVS)

We review the definition of OKVS as introduced in previous work [15]–[18], [20], [43]. In simple terms, an OKVS allows one to encode n key-value pairs in a way that prevents an adversary from recovering the input keys, provided that the values appear random to the adversary. An OKVS is composed of the following two algorithms, defined over a key universe $\mathcal{K}$ and a value universe $\mathcal{V}$.

- $\mathsf{Encode}(\{k_i, v_i\}_{i=1}^n, r)$: The encoding algorithm takes as input n key-value pairs $\{(k_i, v_i)\}_{i=1}^n$ and randomness r, and returns an OKVS $P \in \mathcal{K}^m \cup \{\perp\}$, where $\perp$ denotes an encoding failure.
- $\mathsf{Decode}(P, k, r)$: The decoding algorithm takes as input an OKVS $P \in \mathcal{V}^m$ and a key $k \in \mathcal{K}$, and outputs the corresponding value $v \in \mathcal{V}$.

Encoding success probability. The encoding algorithm fails with negligible probability in the security parameter λ, that is, $\Pr[\mathsf{Encode}(\{(k_i, v_i)\}_{i=1}^n, r) := \perp] < 2^{-\lambda}$.

Decoding correctness. We say that an OKVS satisfies correctness if, for all valid inputs $\{(k_i, v_i)\}_{i=1}^n$, all randomness r, and all $i \in [n]$, the following holds: if $P := \mathsf{Encode}(\{(k_i, v_i)\}_{i=1}^n, r)$, then $\mathsf{Decode}(P, k_i, r) = v_i$. Furthermore, the correctness property does not guarantee the output of Decode for any key $k \notin \{k_1, \ldots, k_n\}$.

Redundancy rate. We define the redundancy rate of an OKVS as $\epsilon := \frac{m}{n}$, where m is the length of the encoded vector and n is the number of input key-value pairs.

Linearity. An OKVS is linear if there exists a function $\mathsf{row} : \mathcal{K} \to \{0,1\}^m$ such that for all $k \in \mathcal{K}$ and $P \in \mathcal{V}^m$,

$$\mathsf{Decode}(P, k, r) = \langle \mathsf{row}(k), P \rangle.$$

Computational indistinguishability. An OKVS is said to be computationally oblivious if, for all pairs of sets of n distinct keys $\{k_1, \ldots, k_n\} \subseteq \mathcal{K}$ and $\{k'_1, \ldots, k'_n\} \subseteq \mathcal{K}$, and for all n values $v_1, \ldots, v_n$ drawn uniformly at random from $\mathcal{V}$, no probabilistic polynomial-time adversary can distinguish between the following two distributions:

- $P := \mathsf{Encode}(\{(k_1, v_1), \cdots, (k_n, v_n)\}, r)$,
- $P' := \mathsf{Encode}(\{(k'_1, v_1), \cdots, (k'_n, v_n)\}, r)$.

Statistical indistinguishability. An OKVS satisfies statistical indistinguishability if, for all sets of n distinct keys $\{k_1, \ldots, k_n\} \subseteq \mathcal{K}$ and uniformly random values $\{v_1, \ldots, v_n\} \subseteq \mathcal{V}$, the distribution of $\mathsf{Encode}(\{(k_1, v_1), \ldots, (k_n, v_n)\}, r)$ is statistically indistinguishable from the uniform distribution over $\mathcal{V}^m$.

Random decodings. An OKVS satisfies random decodings if, for all sets of n distinct keys $K = \{k_1, \ldots, k_n\} \subseteq \mathcal{K}$, and for all n values $v_1, \ldots, v_n$ drawn uniformly at random from $\mathcal{V}$, the output of $\mathsf{Decode}(P, k, r)$ for any key $k \notin K$ is statistically indistinguishable from a uniformly random element in $\mathcal{V}$, where $P := \mathsf{Encode}(\{(k_1, v_1), \ldots, (k_n, v_n)\}, r)$.

D. Cuckoo Hashing

Cuckoo hashing [42] is a data structure that employs k hash functions $h_j : \{0,1\}^* \to [m]$ for determining the positions of n elements. This hashing strategy is widely used for element alignment in PSI. To insert a new element e, cuckoo hashing places it at one of the candidate positions $h_j(e)$ for some $j \in [k]$. If the target location is already occupied by another element e', the existing element e' is evicted and relocated to one of its alternative positions $h_i(e')$, where $i \in [k]$ and $i \neq j$. This process continues recursively until all elements are successfully placed or a pre-defined relocation threshold is reached. If insertion still fails after exhausting the allowed number of evictions, the final element is stored in a stash.

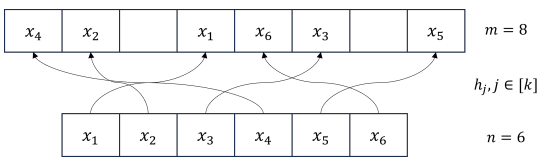

Fig. 13: A simple illustration of Cuckoo hashing with $n = 6$, $m = 8$, and $k = 3$.

Figure 13 illustrates a simple example of cuckoo hashing. The use of cuckoo hashing in PSI aims to avoid the $\mathcal{O}(n^2)$ pairwise comparison of masked elements between the two parties. Instead, it reduces the computation and communication overhead to $\mathcal{O}(m) = \mathcal{O}(n)$ by aligning elements through a set of hash functions $\{h_j\}$. In this work, we also primarily rely on this technique to construct a circuit PSI protocol with linear overhead. The receiver inserts its elements into a cuckoo hash table of length m. Meanwhile, the sender inserts its elements into a simple hashing structure using the same set of hash functions $\{h_j\}$. Specifically, each element is hashed k times and inserted into k positions, where each position can hold an arbitrary number of elements. Both parties perform further evaluation over the two hash tables.

V. Our PSI Protocol

This section presents the construction of the proposed PSI. Note that there exist multiple implementation choices for the underlying OKVS and base random OTs (i.e., public-key operations). Therefore, any improvements to these two components can directly benefit our PSI.

A. Semi-Honest Protocol

Our protocol in the semi-honest model is formally described in Figure 14. The protocol is highly lightweight, requiring only $\log_2(n_s n_r)+\lambda$ base random OTs and a single encoding and n_s decodings of the OKVS. All remaining operations, including the instantiation of the random oracle, can be efficiently implemented using SHA2-NI intrinsics and bitwise operations. In fact, the OKVS encoding and decoding themselves are also realized through highly efficient bitwise computations, making the entire protocol inherently efficient.

Input of $\mathcal{S}$: $X := \{x_1, \ldots, x_{n_s}\}$
Input of $\mathcal{R}$: $Y := \{y_1, \ldots, y_{n_r}\}$
Common: $\mathsf{out} := \ell, \ell := \log_2(n_s n_r) + \lambda, \mathsf{H}^{\mathsf{b}} : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}, \mathsf{H}^\circ : \{0,1\}^{\mathsf{out}} \times \{0,1\}^* \times \{0,1\}^\kappa \to \{0,1\}^{\mathsf{out}}$, $m := \epsilon n_r$
Protocol:

1) $\mathcal{S}$ chooses $s \leftarrow \{0,1\}^\ell, \omega \leftarrow \{0,1\}^\kappa$ uniformly at random.
2) $\mathcal{S}$ and $\mathcal{R}$ invoke ℓ instances of $\mathcal{F}_{\mathsf{ROT}}^\ell$. In the i-th instance:
   - $\mathcal{S}$ acts as receiver with input $s[i]$.
   - $\mathcal{R}$ acts as the sender and receives $a_i, b_i \in \{0,1\}^\kappa$.
   - $\mathcal{S}$ receives output c_i.
3) For $y \in Y$, $\mathcal{R}$ computes $D(y) := A(y) \oplus B(y)$, where:
$$A(y) := ||_{i=1}^{\ell} \mathsf{H}^{\mathsf{b}}(a_i, y), B(y) := ||_{i=1}^{\ell} \mathsf{H}^{\mathsf{b}}(b_i, y)$$
4) $\mathcal{R}$ chooses $r \leftarrow \{0,1\}^\kappa$ uniformly at random, encodes an OKVS $P \in \mathbb{F}_{2^\ell}^m$ using $\mathsf{Encode}(\{y, D(y)\}, r)$ for $y \in Y$, and sends $\{r, P\}$ to $\mathcal{S}$.
5) $\mathcal{S}$ defines $C(x) := \|_{i=1}^{\ell} \mathsf{H}^{\mathsf{b}}(c_i, x)$ and sends $\omega, X' := \{\mathsf{H}^\circ(C(x) \oplus (s \wedge \mathsf{Decode}(P, x, r)), x, \omega)\}$ for $x \in X$ (randomly permuted) to $\mathcal{R}$.
6) $\mathcal{R}$ computes $Y' := \{\mathsf{H}^\circ(A(y), y, \omega)\}$ for $y \in Y$ and outputs $Z = X \cap Y$ based on $X' \cap Y'$.

Fig. 14: Our semi-honest PSI protocol.

Since the output length of H° is set to $\log_2(n_s n_r) + \lambda$, the probability of a collision is $\mathcal{O}(2^{-\lambda})$. Assuming no collisions occur in H°, we briefly outline the correctness of the protocol. It is necessary to verify whether $C(x) \oplus (s \wedge \mathsf{Decode}(P, x, r)) = A(y)$ when $x = y$. If $y = x$, we have:

$$\begin{aligned} &C(y) \oplus (s \wedge \mathsf{Decode}(P, y, r)) \\ &\quad = ||_{i=1}^{\ell} \mathsf{H}^{\mathsf{b}}(c_i, y) \oplus (s[i](\mathsf{H}^{\mathsf{b}}(a_i, y) \oplus \mathsf{H}^{\mathsf{b}}(b_i, y))) \\ &\quad = ||_{i=1}^{\ell} \mathsf{H}^{\mathsf{b}}(a_i, y) = A(y). \end{aligned}$$

A straightforward verification shows that the equality holds for both cases of $s[i] \in \{0,1\}$, thereby establishing the correctness of the protocol.

Theorem 1. *Our PSI protocol, as illustrated in Figure 14, realizes the $\mathcal{F}_{\mathsf{PSI}}$ functionality in Figure 8 against a semi-honest adversary.*

Proof. Corrupted $\mathcal{S}$. The view of $\mathcal{S}$ in the protocol described in Figure 14 consists of $\{s, \omega, \{c_i\}, r, P\}$. The simulator interacts with $\mathcal{S}$ as follows:

- $\mathcal{S}$ samples $s \leftarrow \{0,1\}^\ell, \omega \leftarrow \{0,1\}^\kappa$ uniformly at random.
- The simulator emulates $\mathcal{F}_{\mathsf{ROT}}^\ell$, waits for $\mathcal{S}$ to send s and the corresponding response $\{c_i\}$.
- $\mathcal{S}$ chooses $r \leftarrow \{0,1\}^\kappa$ uniformly at random.
- On behalf of $\mathcal{R}$, the simulator samples $P \leftarrow \mathbb{F}_{2^\ell}^m$ uniformly at random and sends it to $\mathcal{S}$.

This simulation is indistinguishable from the real world by the following hybrids:

- $\mathcal{H}_0$. The same as the real protocol with an honest receiver $\mathcal{R}$ and $\mathcal{F}_{\mathsf{ROT}}^\ell$ is executed honestly.
- $\mathcal{H}_1$. The same as $\mathcal{H}_0$ except the simulator emulates $\mathcal{F}_{\mathsf{ROT}}^\ell$.
- $\mathcal{H}_2$. The same as in $\mathcal{H}_1$, since r is also chosen uniformly at random in the real protocol.
- $\mathcal{H}_3$. The simulator aborts if H^{b} has been previously queried on a_i when $s[i] = 1$ or on b_i when $s[i] = 0$. The probability of this occurring is $\mathcal{O}(2^{-\kappa})$.
- $\mathcal{H}_4$. The simulator in this hybrid does not call Encode, and so does not abort if Encode fails. Same as in $\mathcal{H}_3$, except that a randomly sampled $P \leftarrow \mathbb{F}_{2^\ell}^m$ is used in place of the honestly generated OKVS in the real protocol. In the real protocol, P is generated as $\mathsf{Encode}(\{y, D(y)\}, r)$. To argue that an honestly generated OKVS is indistinguishable from a uniformly random P, it is crucial to show that $D(y)$ is uniformly random from the view of $\mathcal{S}$. Since $D(y) = A(y) \oplus B(y)$, where $A(y) = \|_{i=1}^{\ell} \mathsf{H}^{\mathsf{b}}(a_i, y)$ and $B(y) = \|_{i=1}^{\ell} \mathsf{H}^{\mathsf{b}}(b_i, y)$, it can be observed that each bit of $D(y)$ is indistinguishable to $\mathcal{S}$. Specifically, $\mathcal{S}$ only learns $c_i = a_i$ or b_i depending on s, so each bit of $D(y)$ appears uniformly random over $\mathbb{F}_2$, as either $A(y)$ or $B(y)$ is uniformly random. Therefore, by the computational obliviousness and double obliviousness of the OKVS, this hybrid is computationally indistinguishable.

Corrupted $\mathcal{R}$. The view of $\mathcal{R}$ in the protocol described in Figure 14 consists of $\{\{a_i\}, \{b_i\}, \omega, X'\}$. Unlike the case with a corrupted $\mathcal{S}$, it is necessary to ensure the correctness

here. The simulator takes as input the receiver's set Y, the intersection $Z = X \cap Y$, and plays the role of $\mathcal{F}_{\mathsf{ROT}}^{\ell}$. The simulator interacts with $\mathcal{R}$ as follows:

- The simulator samples $\{a_i\}$ and $\{b_i\}$ uniformly at random from $\{0,1\}^{\kappa}$.
- The simulator emulates $\mathcal{F}_{\mathsf{ROT}}^{\ell}$, sends $\{a_i\}, \{b_i\}$ to $\mathcal{R}$.
- On behalf of $\mathcal{S}$, the simulator chooses $s \leftarrow \{0,1\}^{\ell}$ uniformly at random and sets $c_i := a_i$ if $s[i] = 0$; otherwise, $c_i := b_i$.
- The simulator chooses $\omega \leftarrow \{0,1\}^{\kappa}$ uniformly at random.
- The simulator computes Y' and generates a uniformly random X' such that $X' \cap Y' = \{\mathsf{H}^{\circ}(A(z), z, \omega) \mid z \in Z\}$.

To demonstrate the correctness of this simulation and its indistinguishability from the real protocol, consider the following hybrids:

- $\mathcal{H}_0$. The same as the real protocol with an honest sender $\mathcal{S}$ and $\mathcal{F}_{\mathsf{ROT}}^{\ell}$ is executed honestly.
- $\mathcal{H}_1$. The same as $\mathcal{H}_0$ except the simulator emulates $\mathcal{F}_{\mathsf{ROT}}^{\ell}$, receives $\{a_i\}, \{b_i\}$ from $\mathcal{R}$.
- $\mathcal{H}_2$. The simulator aborts if Encode fails, which occurs with probability $2^{-\lambda}$.
- $\mathcal{H}_3$. For the items in the intersection $z \in Z = X \cap Y$, we have

$$\begin{aligned} &C(z) \oplus (s \wedge \mathsf{Decode}(P, z, r)) \\ &\quad = ||_{i=1}^{\ell} \mathsf{H}^{\mathsf{b}}(c_i, z) \oplus (s[i](\mathsf{H}^{\mathsf{b}}(a_i, z) \oplus \mathsf{H}^{\mathsf{b}}(b_i, z))) \\ &\quad = ||_{i=1}^{\ell} \mathsf{H}^{\mathsf{b}}(a_i, z) \end{aligned}$$

  regardless of whether $s[i]$ is 0 or 1. Therefore, for all $y \in Z$, $\mathcal{R}$ can indeed find a matching element in X'. Now consider the items of X not in the intersection, $X^* = X \backslash Y$. For $x^* \in X^*, y \in Y$, correctness of the protocol requires

$$\begin{aligned} &C(x^*) \oplus (s \wedge \mathsf{Decode}(P, x^*, r)) \neq A(y) \\ &||_{i=1}^{\ell} \mathsf{H}^{\mathsf{b}}(c_i, x^*) \oplus (s[i](\mathsf{H}^{\mathsf{b}}(a_i, x^*) \oplus \mathsf{H}^{\mathsf{b}}(b_i, x^*))) \\ &\neq ||_{i=1}^{\ell} \mathsf{H}^{\mathsf{b}}(a_i, y). \end{aligned}$$

  The probability of this case causing an abort is $2^{-\lambda}$, since $\ell = \log_2(n_s n_r) + \lambda$.
- $\mathcal{H}_4$. This hybrid aborts if any $(x^*, y) \in X^* \times Y$ results in a collision in the corresponding H° queries. The probability of this event occurring is also $2^{-\lambda}$.
- $\mathcal{H}_5$. This hybrid aborts if $\mathcal{R}$ ever queries $\mathsf{H}^{\circ}(\cdot, x^*, \omega)$ at $C(x^*) \oplus (s \wedge \mathsf{Decode}(P, x^*, r))$ for any $x^* \in X^*$. Since ω is uniformly distributed with respect to $\mathcal{R}$ prior to being sent, the probability of this event occurring is $2^{-\kappa}$.

This completes the proof of the protocol in the semi-honest model. □

B. Maliciously Secure Protocol

Our PSI can be extended from the semi-honest to the malicious model using a standard transformation, which has been employed in several prior works [15]–[18], [56]. Our malicious-secure PSI protocol is formally described in Figure 15. For a detailed discussion on certain parameter choices, such as the rationale for setting $\mathsf{out} := \kappa$, see [15], [18].

Input of $\mathcal{S}$: $X := \{x_1, \ldots, x_{n_s}\}$
Input of $\mathcal{R}$: $Y := \{y_1, \ldots, y_{n_r}\}$
Common: $\mathsf{out} = \kappa, \mathsf{H}^{\mathsf{b}} : \{0,1\}^{\kappa} \times \{0,1\}^{*} \times \{0,1\}^{\kappa} \to \{0,1\}, \mathsf{H}^{\kappa} : \{0,1\}^{*} \to \{0,1\}^{\kappa}, \mathsf{H}^{\circ} : \{0,1\}^{\mathsf{out}} \times \{0,1\}^{*} \to \{0,1\}^{\mathsf{out}}$, $m := \epsilon n_r$
Protocol:

1) $\mathcal{S}$ chooses $s, \omega_1, \tau \leftarrow \{0,1\}^{\kappa}$ uniformly at random and sends $\tau_1 := \mathsf{H}^{\kappa}(\omega_1), \tau$ to $\mathcal{R}$.
2) $\mathcal{S}$ and $\mathcal{R}$ invoke κ instances of $\mathcal{F}_{\mathsf{ROT}}^{\kappa}$. In the i-th instance:
   - $\mathcal{S}$ acts as receiver with input $s[i]$.
   - $\mathcal{R}$ acts as the sender and receives $a_i, b_i \in \{0,1\}^{\kappa}$.
   - $\mathcal{S}$ receives output c_i.
3) For $y \in Y$, $\mathcal{R}$ computes $D(y) := A(y) \oplus B(y)$, where:
$$A(y) := ||_{i=1}^{\kappa} \mathsf{H}^{\mathsf{b}}(a_i, y, \tau), B(y) := ||_{i=1}^{\kappa} \mathsf{H}^{\mathsf{b}}(b_i, y, \tau)$$
4) $\mathcal{R}$ chooses $r, \omega_2 \leftarrow \{0,1\}^{\kappa}$ uniformly at random, encodes an OKVS $P \in \mathbb{F}_{2^{\kappa}}^{m}$ using $\mathsf{Encode}(\{y, D(y)\}, r)$ for $y \in Y$, and sends $\{r, P, \omega_2\}$ to $\mathcal{S}$.
5) $\mathcal{S}$ sends ω_1 to $\mathcal{R}$ who aborts if $\tau_1 \neq \mathsf{H}^{\kappa}(\omega_1)$. Both parties define $\omega := \omega_1 \oplus \omega_2$.
6) $\mathcal{S}$ defines $C(x) := ||_{i=1}^{\kappa} \mathsf{H}^{\mathsf{b}}(c_i, x, \tau)$ and sends $X' := \{\mathsf{H}^{\circ}(C(x) \oplus (s \wedge \mathsf{Decode}(P, x, r)) \oplus \omega, x)\}$ for $x \in X$ (randomly permuted) to $\mathcal{R}$.
7) $\mathcal{R}$ computes $Y' := \{\mathsf{H}^{\circ}(A(y) \oplus \omega, y)\}$ for $y \in Y$ and outputs $Z = X \cap Y$ based on $X' \cap Y'$.

Fig. 15: Our maliciously secure PSI protocol.

The correctness of the protocol has already been discussed in the semi-honest setting. Moreover, correctness is implicitly guaranteed in the malicious model, as it is necessary for achieving indistinguishability between the real and ideal worlds. Therefore, we proceed directly to the security analysis.

Theorem 2. *Our PSI protocol, as illustrated in Figure 15, realizes the $\mathcal{F}_{\mathsf{PSI}}$ functionality in Figure 8 against a malicious adversary.*

Proof. Corrupted $\mathcal{S}$. Consider a malicious $\mathcal{S}$. The simulator interacts with $\mathcal{S}$ as follows:

- The simulator emulates the functionality $\mathcal{F}_{\mathsf{ROT}}^{\kappa}$ and waits for party $\mathcal{S}$ to send $s \in \{0,1\}^{\kappa}$. It then samples random strings $\{(a_i, b_i)\}_{i=1}^{\kappa}$, and returns a_i if $s[i] = 0$, or b_i otherwise.
- On behalf of $\mathcal{R}$, the simulator sends uniform $\{r, P, \omega_2\}$ to $\mathcal{S}$.
- Let X^* denote the set of all elements x that are queried to H°. When $\mathcal{S}$ sends X', the simulator computes $\hat{X} := \{x \mid x \in X^* \wedge \nexists x' \in X^* \text{ s.t. } x \neq x' \wedge \mathsf{H}^{\circ}(\cdot, x) = \mathsf{H}^{\circ}(\cdot, x')\}$, extracts $X := \{x \mid x \in \hat{X} \wedge \mathsf{H}^{\circ}(\cdot, x) \in X'\}$, and sends X to $\mathcal{F}_{\mathsf{PSI}}$.

To prove that this simulation is indistinguishable and consider the following hybrids:

- $\mathcal{H}_0$. The same as the real protocol except the simulator in this hybrid plays the role of $\mathcal{F}_{\mathsf{ROT}}^{\kappa}$.

- $\mathcal{H}_1$. The simulator aborts if H^{b} has been previously queried on a_i when $s[i] = 1$ or on b_i when $s[i] = 0$. The probability of this occurring is $\mathcal{O}(2^{-\kappa})$.
- $\mathcal{H}_2$. In this hybrid, the simulator aborts when sampling r if there exists a prior query to $\mathsf{row}(\cdot, r)$. Since r is sampled uniformly at random, the probability of this case is $\mathcal{O}(2^{-\kappa})$, and therefore this hybrid is indistinguishable from $\mathcal{H}_1$.
- $\mathcal{H}_3$. In this hybrid, the simulator samples $P \leftarrow \mathbb{F}_{2^\kappa}^m$ uniformly as opposed to $P := \mathsf{Encode}(\{y, D(y)\}, r)$. Therefore, it does not abort due to encoding failure. Since no prior query to $\mathsf{row}(\cdot, r)$ has been made and the H° query does not result in an abort, the indistinguishability of the OKVS holds: because the values are random from the perspective of $\mathcal{S}$, the resulting OKVS P appears uniformly random to $\mathcal{S}$. Hence, this hybrid is indistinguishable from $\mathcal{H}_2$.
- $\mathcal{H}_4$. Whenever $\mathcal{S}$, after receiving P, queries $\mathsf{H}^\circ(q, x)$, where $q = C(x) \oplus (s \wedge \mathsf{Decode}(P, x, r)) \oplus \omega$, and $\mathsf{H}^\circ(q, x)$ has already been queried before, the hybrid aborts. Otherwise, the simulator responds using uniformly sampled elements. Note that in this hybrid, r and ω are uniformly distributed before r and ω_2 are sent. Therefore, any given $q = C(x) \oplus (s \wedge \mathsf{Decode}(P, x, r)) \oplus \omega$ is uniformly distributed, and the probability that $\mathcal{S}$ has previously queried $\mathsf{H}^\circ(q, x)$ is negligible. Therefore, this hybrid is indistinguishable from the previous one.
- $\mathcal{H}_5$. Assuming no collisions of the form $\mathsf{H}^\circ(\cdot, x) = \mathsf{H}^\circ(\cdot, x')$, the correctness and indistinguishability of the simulation follow directly. The main case that requires discussion is when collisions do occur. Note that the simulator only needs to extract x and x' if at least one of them appears in Y with non-negligible probability. Without loss of generality, assume $x \in Y$. Consider some $y \in Y$ and the probability that $\mathsf{H}^\circ(\cdot, x') = \mathsf{H}^\circ(\cdot, y)$. Since $|Y| = \mathcal{O}(\kappa)$, the probability that $\mathcal{S}$ finds such a target collision is $\mathcal{O}(2^{-\kappa})$.

Corrupted $\mathcal{R}$. Consider a malicious $\mathcal{R}$. The simulator is as follows:

- The simulator plays the role of $\mathcal{F}_{\mathsf{ROT}}^{\kappa}$.
- The simulator generates the maskings of Y, denoted Y', uniformly at random.
- The simulator forwards Y to $\mathcal{F}_{\mathsf{PSI}}$ and receives $Z = X \cap Y$ in response.
- The simulator generates X' by randomly selecting $|Z|$ elements from Y' and $n_s - |Z|$ elements from $\{0,1\}^\kappa \setminus Y'$.
- The simulator sends X' to $\mathcal{R}$.

To prove that the simulation is indistinguishable, we consider the following hybrids:

- $\mathcal{H}_0$. This hybrid is identical to the real protocol, except that the simulator plays the role of $\mathcal{F}_{\mathsf{ROT}}^{\kappa}$ and samples $\{a_i\}$ and $\{b_i\}$.
- $\mathcal{H}_1$. If a query to H^{b} involving τ is made before τ has been sent, the simulator aborts. The probability of this case occurring is $\mathcal{O}(2^{-\kappa})$.
- $\mathcal{H}_2$. After $\mathcal{R}$ sends r and P, the hybrid checks, for all prior H^{b} queries made by $\mathcal{R}$, whether the condition $\mathsf{Decode}(P, y, r) = ||_{i=1}^{\kappa} \mathsf{H}^{\mathsf{b}}(a_i, y, \tau) \oplus \mathsf{H}^{\mathsf{b}}(b_i, y, \tau)$ holds. If the condition holds, then y is added to the set Y. In this hybrid, the simulator generates the maskings of Y, denoted Y', uniformly at random.
- $\mathcal{H}_3$. In this hybrid, the simulator does not sample ω_1 at the beginning of the protocol. Instead, it sends a random value as τ_1 in place of $\mathsf{H}^\kappa(\omega_1)$. When the point in the protocol is reached where ω_1 should be sent, the simulator samples ω_1 and sets $\mathsf{H}^\kappa(\omega_1) := \tau_1$, under the condition that ω_1 has not been queried before. In this case, the distribution of the hybrid remains identical to the previous one and is thus indistinguishable, since ω_1 is uniformly distributed.
- $\mathcal{H}_4$. After $\omega_1 \leftarrow \{0,1\}^\kappa$ is sampled, if $\mathcal{R}$ has previously issued a query of the form $\mathsf{H}^\circ(A(y) \oplus \omega, y)$ for all $y \in Y$ after receiving P, then the simulator aborts. Since ω_1 is freshly sampled, each value of $A(y) \oplus \omega$ is uniformly random, and thus the probability of aborting is $\mathcal{O}(2^{-\kappa})$. In this hybrid, the simulator programs H° for all $y \in Y$ such that $\{\mathsf{H}^\circ(A(y) \oplus \omega, y) \in Y'\}$, and sends ω_2 to $\mathcal{R}$. Since Y' is uniformly distributed, programming H° in this way does not alter the distribution.
- $\mathcal{H}_5$. If $\mathcal{R}$ has previously made a query of the form $\mathsf{H}^\circ(q, y)$ where $(q, y) \in \{(C(y) \oplus (s \wedge \mathsf{Decode}(P, y, r)) \oplus \omega, y) \mid y \in \{0,1\}^\kappa \setminus Y\}$, then the simulator aborts. From the view of $\mathcal{R}$, the s is uniformly distributed. Therefore, the probability that this causes an abort is $\mathcal{O}(2^{-\kappa})$.
- $\mathcal{H}_6$. In this hybrid, the only difference between the simulated X' and that in the real protocol is that, in the real protocol, the dummy elements are sampled uniformly from the entire set $\{0,1\}^{\mathsf{out}}$, whereas in the simulation, they are sampled from $\{0,1\}^{\mathsf{out}} \setminus Y'$. However, $\{0,1\}^{\mathsf{out}}$ is extremely large (of size 2^κ), while $|X|$ is negligible in comparison. Therefore, $|\{0,1\}^{\mathsf{out}} \setminus Y'| = 2^{\mathsf{out}} - |X| = \mathcal{O}(2^\kappa)$, making the hybrid indistinguishable.

This completes the proof of the protocol in the malicious model. □

C. Resistance to OKVS overfitting attack

Since the receiver can encode more than n_r key-value pairs into the OKVS, potentially even exceeding the upper bound n_r' allowed by the ideal functionality, this phenomenon is known as the overfitting problem of OKVS. Han et al. [18] formalized this attack, demonstrated that the VOLE+OKVS-based PSI [15] is vulnerable to this issue, and proposed an effective countermeasure. Following their approach, we show that our PSI protocol is also resistant to this attack. In the semi-honest setting, [18] recommended setting the mask output length out to κ, rather than the typical value of $\lambda + \log_2(n_s n_r)$. The reason is that the receiver may issue q queries to the random oracle after the protocol execution, potentially resulting in approximately $\frac{q}{2^{\mathsf{out}}}$ additional evaluations. Although the insight is novel and well-founded, $\lambda + \log_2(n_s n_r)$ is retained for out. Since in the semi-honest model the receiver is assumed

to follow the protocol faithfully and encode exactly n_r key-value pairs into the OKVS during execution, any additional encoding or evaluation would constitute malicious behavior, which falls outside the assumptions of the semi-honest setting. Furthermore, since an output length of $\lambda + \log_2(n_s n_r)$ is adopted by nearly all mainstream semi-honest PSI protocols [10]–[14], [26], it is reasonable to adhere to the same setting in our design. In the malicious setting, an adversary may precompute random oracle queries, thereby enabling the execution of the overfitting attack. Following the approach in [18], a salt value τ, sampled by the sender, is incorporated into H^{b} to prevent such scenarios. In addition, [18] proposed a timeout mechanism to prevent a malicious receiver from issuing excessive H^{b} queries after learning τ in an attempt to carry out the overfitting attack. However, this mechanism cannot be effectively implemented in practice, as it is infeasible to specify a fixed timeout value. Meanwhile, any reasonable estimate for the timeout would likely exceed the expected execution time of PSI protocols in real-world deployments and would also be significantly longer than typical channel latency in practical systems. In fact, [18] does not explicitly specify this timeout in their protocol, merely suggesting that such a mechanism could mitigate random oracle queries issued during the interval between the sender's transmission of τ and the receiver's generation of the OKVS. This consideration is equally applicable to our PSI.

D. Theoretical Analysis

As noted in [15], it is generally challenging to compare the computational overhead of modern efficient PSI protocols through theoretical analysis, since most operations rely on highly optimized symmetric-key operations. Consequently, the practical performance of PSI protocols can only be assessed through empirical evaluation, which is presented in the experimental section (Section VII). Here, we focus primarily on analyzing the communication overhead.

TABLE II: Theoretical communication cost comparison under comparable settings between our PSI protocol and the state-of-the-art OTE-based and VOLE-based PSI protocols in both semi-honest and malicious models. We set $\kappa = 128$, $\lambda = 40$, and the OKVS redundancy rate $\epsilon = 1.03$ [17], which is the lowest known value for efficient OKVS construction.

Protocol	Communication	$n = n_s = n_r$		
		2^{16}	2^{20}	2^{24}
Semi-Honest				
KKRT [12]	$6\kappa n_r + 3(\lambda + \log(n_s n_r))n_s$	$984n$	$1008n$	$1032n$
VOLE-based PSI [15]–[17]	$(\log(n_s n_r) + \lambda)(\epsilon n_r + n_s) + 2^{15.2}\kappa$	$220n$	$167n$	$179n$
Ours	$(\log(n_s n_r) + \lambda)(\epsilon n_r + n_s) + 2^{8}\kappa$	$147n$	$163n$	$179n$
Malicious				
VOLE-based PSI [15]–[17]	$\epsilon\kappa n_r + \kappa n_s + 2^{15.2}\kappa$	$334n$	$265n$	$261n$
Ours	$\epsilon\kappa n_r + \kappa n_s + 2^{8}\kappa$	$261n$	$260n$	$260n$

We compare the theoretical communication costs of our PSI protocol with those of the most efficient OTE-based PSI [12] and VOLE-based PSI protocols [15]–[17] in Table II. The communication costs of silent VOLE [29] and base OTs [57] are measured to be $4{,}715{,}288$ bits and $33{,}024$ bits, respectively, in our practical evaluation. Simplifying these results, the communication costs of VOLE and κ base OTs can be approximated as $2^{15.2}\kappa$ and $2^{8}\kappa$ bits, respectively. Compared to KKRT [12], our PSI achieves a significant reduction in communication cost. While our protocol also outperforms VOLE-based PSI protocols [15]–[17] in terms of communication, the improvement is relatively modest. However, we can show that under the same OKVS redundancy and within the same security model, the communication of our protocol is always smaller than that of VOLE-based PSI.

Theorem 3. *When the OKVS redundancy parameter ϵ is the same and the protocols operate under the same security model, the communication cost of our PSI protocol is strictly smaller than that of the VOLE-based PSI.*

Proof. For the OKVS part, both our PSI and the VOLE-based PSI require $\epsilon\ell n_r$ bits of communication, where in the semi-honest model $\ell = \log(n_s n_r) + \lambda$, and in the malicious model $\ell = \kappa$. For the masks sent from the sender to the receiver, the communication cost amounts to ℓn_s bits. Let the communication costs of our PSI and the VOLE-based PSI be denoted as C_{our} and C_{volepsi}, respectively. Therefore, we have

$$C_{\mathsf{our}} = \epsilon\ell n_r + \ell n_s + C_{\mathsf{baseot}},$$
$$C_{\mathsf{volepsi}} = \epsilon\ell n_r + \ell n_s + C_{\mathsf{vole}},$$

where C_{baseot} denotes the communication of κ base OTs and C_{vole} that of the VOLE protocol. Since VOLE inherently requires base random OTs, we have $C_{\mathsf{baseot}} \leq C_{\mathsf{vole}}$. Therefore, under identical settings, we conclude that $C_{\mathsf{our}} \leq C_{\mathsf{volepsi}}$. □

VI. Extension to Circuit PSI

This section extends the PSI protocol to circuit PSI and subsequently to various PSI variants. This requires the use of cuckoo hashing, which makes it difficult to achieve malicious security. For details, refer to the work of Pinkas et al. [20]. Therefore, all protocols in this section are considered under the semi-honest model only. Note that most mainstream circuit-based PSI protocols [15]–[17], [22]–[24] achieve also only semi-honest security. The only known maliciously secure construction [25] remains impractical, running nearly $20\times$ slower than [23].

Our circuit PSI protocol is formally described in Figure 16. Using cuckoo hashing on one side and simple hashing on the other enables bin-to-bin comparison, reducing the matching complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ [15], [22]–[24]. We employ stash-free cuckoo hashing since when $k = 3$ and $m = 1.27n_r$, the probability of insertion failure is $< 2^{-40}$ and thus negligible [22]. If there is an intersection at position $i \in [m]$, the receiver obtains $r'_i = r_i$ and $\tau'_i = t_{\tilde{x}} \oplus \tau_i$. This is because when an intersection occurs ($t'_x = t'_y$), we have $t'_x \oplus (r_i \| t_{\tilde{x}} \oplus \tau_i) \oplus t'_y = (r_i \| t_{\tilde{x}} \oplus \tau_i) = r'_i \| \tau'_i$. Therefore, $\mathcal{S}$ holds $(r_i \| \tau_i)$, and $\mathcal{R}$ sends $(r'_i \| \tau'_i, t_{\tilde{y}})$, after which they invoke $\mathcal{F}_{\mathsf{2PC}}$ to compute whether $r_i = r'_i$. If equality holds, it indicates an intersection, i.e., $q_i = 1$, and in this case $z_i = (t_{\tilde{x}} \| t_{\tilde{y}})$. The shares of q_i and z_i are then used for the subsequent circuit evaluation. Note that the length of r_i must be at least $\log_2 m + \lambda$ bits to ensure a collision probability of $\mathcal{O}(2^{-\lambda})$ over m comparisons [22].

Input of $\mathcal{S}$: $X := \{x_1, \ldots, x_{n_s}\}$ and $\tilde{X} := \{\tilde{x}_1, \ldots, \tilde{x}_{n_s}\}$
Input of $\mathcal{R}$: $Y := \{y_1, \ldots, y_{n_r}\}$ and $\tilde{Y} := \{\tilde{y}_1, \ldots, \tilde{y}_{n_r}\}$
Common: Cuckoo hash table size $m := 1.27n_r$, and $k := 3$ hash functions $h_j : \{0,1\}^* \to [m]$, $\mathsf{H} : \{0,1\}^* \times \{0,1\}^\kappa \to \{0,1\}^\kappa$.
Protocol:

1) $\mathcal{S}$ samples $r \leftarrow \{0,1\}^\kappa$ uniformly at random and sends it to $\mathcal{R}$.
2) $\mathcal{R}$ constructs a cuckoo hash table T_Y of Y such that for each $y \in Y$, there exists $j \in [k]$ such that $\mathsf{H}(y\|j, r) = T_Y[h_j(y)]$. The empty entries of T_Y are filled with random elements.
3) $\mathcal{S}$ constructs a simple hash table T_X of X such that for each $x \in X$ and $j \in [k]$, it holds that $\mathsf{H}(x\|j, r) \in T_X[h_j(x)]$. For each $t_x \in T_X$, its corresponding associated value $\tilde{x}$ is denoted as $t_{\tilde{x}} \in T_{\tilde{X}}$.
4) Both parties run our PSI with T_X and T_Y as inputs, except that $\mathcal{S}$ omits sending the resulting masked set T'_X to $\mathcal{R}$. $\mathcal{S}$ and $\mathcal{R}$ obtain the masked sets T'_X and T'_Y corresponding to T_X and T_Y, respectively. In other words, the execution of Figure 14 terminates at Step 5, and the sender does not send the corresponding masked set T'_X.
5) For all $i \in [m]$, $\mathcal{S}$ samples $r_i \leftarrow \{0,1\}^{\log_2 m + \lambda}, \tau_i \leftarrow \{0,1\}^{|\tilde{x}|}$ uniformly at random.
6) $\mathcal{S}$ encodes an OKVS P using $\mathsf{Encode}(\{(t_x, t'_x \oplus (r_i\|t_{\tilde{x}} \oplus \tau_i))\}, r)$ for each $t_x \in T_X$, where $t'_x \in T'_X$ and $t_{\tilde{x}} \in T_{\tilde{X}}$ are the PSI-generated mask and the associated value corresponding to t_x, respectively, and sends P to $\mathcal{R}$.
7) For each $i \in [m]$, $\mathcal{R}$ computes $r'_i\|\tau'_i := \mathsf{Decode}(P, t_y, r) \oplus t'_y$, where t_y is the value at position i in T_Y, and t'_y is the corresponding PSI-generated mask.
8) For each $i \in [m]$, $\mathcal{S}$ sends $r_i\|\tau_i$, and $\mathcal{R}$ sends $(r'_i\|\tau'_i, t_{\tilde{y}})$ to $\mathcal{F}_{\mathsf{2PC}}$, where $t_{\tilde{y}}$ is the associated value corresponding to the i-th position in T_Y (or zero if this is a dummy entry). $\mathcal{F}_{\mathsf{2PC}}$ computes a circuit that, for each $i \in [m]$, sets $q_i := 1$ if $r_i = r'_i$, and $q_i := 0$ otherwise, and outputs secret shares (q_i^0, q_i^1) of q_i, and (z_i^0, z_i^1) of $z_i := q_i \cdot ((\tau'_i \oplus \tau_i)\|t_{\tilde{y}})$.

Fig. 16: Our circuit PSI protocol.

Moreover, this implies that $\mathcal{R}$ obtains a secret share of the payload corresponding to each intersection, as $t_{\tilde{x}} = \tau'_i \oplus \tau_i$. Consequently, regardless of how many elements are inserted into each position of T_X, only one evaluation per position is needed during circuit computation. This strictly guarantees that both the computational and communication complexities remain $\mathcal{O}(m) = \mathcal{O}(n)$.

Theorem 4. *Our circuit PSI protocol, as illustrated in Figure 16, realizes the $\mathcal{F}_{\mathsf{CPSI}}$ functionality in Figure 9 against a semi-honest adversary.*

Proof. Corrupted $\mathcal{S}$. In the circuit PSI protocol, the view of $\mathcal{S}$ is $\{r, \{r_i\}, \{\tau_i\}, Q^1, Z^1\}$, excluding the interaction messages of the PSI protocol. Simulating this view is trivial: Q^1 and Z^1 can be obtained by invoking $\mathcal{F}_{\mathsf{2PC}}$, while all other components are random values. Moreover, the interaction messages of the PSI protocol have already been proven to be simulatable.

Corrupted $\mathcal{R}$. The view of $\mathcal{R}$ is $\{r, P, Q^0, Z^0\}$, excluding the interaction messages of the PSI protocol. The primary consideration is whether P can be correctly simulated. According to the indistinguishability property of OKVS, for P to be indistinguishable to $\mathcal{R}$, the value $t'_x \oplus (r_i\|t_{\tilde{x}} \oplus \tau_i)$ should appear random to $\mathcal{R}$. We can observe that t'_x is uniformly random from the perspective of $\mathcal{R}$, since it is in fact the mask of T_X generated by the PSI protocol. Therefore, P is indistinguishable from the perspective of $\mathcal{R}$. The proofs for the remaining interaction messages are straightforward. □

A. Extending Circuit PSI to Additional Functionalities

Circuit PSI serves as a general framework for realizing various PSI variants. By adapting the evaluation circuit in the final step, different functionalities can be naturally supported. In our work, we have extended it to support PSI-cardinality, PSI-sum, and PJC.

PSI-cardinality. The PSI-cardinality protocol is described formally in Figure 17. As no payloads are involved in PSI-cardinality, the operations on $\tilde{X}$ and $\tilde{Y}$ are removed accordingly. The final circuit computes $\mathsf{ca} := \sum_{i=1}^m q_i$, where q_i denotes the presence of an intersection at position i. By directly outputting the intersection cardinality ca without leaking any intermediate information, the functionality of PSI-cardinality is realized.

Input of $\mathcal{S}$: $X := \{x_1, \ldots, x_{n_s}\}$
Input of $\mathcal{R}$: $Y := \{y_1, \ldots, y_{n_r}\}$
Common: $m := 1.27n_r$, and $k := 3$ hash functions $h_j : \{0,1\}^* \to [m]$, $\mathsf{H} : \{0,1\}^* \times \{0,1\}^\kappa \to \{0,1\}^\kappa$.
Protocol:

1) Steps 1 through 4 are consistent with those of the circuit PSI protocol (Figure 16).
2) For all $i \in [m]$, $\mathcal{S}$ samples $r_i \leftarrow \{0,1\}^{\log_2 m + \lambda}$ uniformly at random.
3) $\mathcal{S}$ encodes an OKVS P using $\mathsf{Encode}(\{(t_x, t'_x \oplus r_i)\}, r)$ for $t_x \in T_X$ and sends P to $\mathcal{R}$.
4) For each $i \in [m]$, $\mathcal{R}$ computes $r'_i := \mathsf{Decode}(P, t_y, r) \oplus t'_y$.
5) For each $i \in [m]$, $\mathcal{S}$ sends r_i and $\mathcal{R}$ sends r'_i to $\mathcal{F}_{\mathsf{2PC}}$, which evaluates a circuit that sets $q_i := 1$ if $r_i = r'_i$, and $q_i := 0$ otherwise, and then computes $\mathsf{ca} := \sum_{i=1}^m q_i$, outputting the result to $\mathcal{R}$.

Fig. 17: Our PSI-cardinality protocol.

Theorem 5. *Our PSI-cardinality protocol, as illustrated in Figure 17, realizes the $\mathcal{F}_{\mathsf{PSICA}}$ functionality in Figure 10 against a semi-honest adversary.*

Proof. The proof is trivial and follows similarly to the proof of Theorem 4. □

PSI-sum. In PSI-sum, only the sender $\mathcal{S}$ is associated with payloads corresponding to its input elements, whereas the receiver $\mathcal{R}$ has no associated payloads. The PSI-sum protocol is described formally in Figure 18. The circuit for PSI-sum computes $\mathsf{sum} := \sum_{i=1}^m q_i(\tau'_i \oplus \tau_i)$. It can be verified that when $q_i = 1$, the value $\tau'_i \oplus \tau_i$ equals the associated payload $\tilde{x}$. From this, $\mathsf{sum} = \sum_{x \in X \cap Y} \tilde{x}$, which realizes PSI-sum.

Theorem 6. *Our PSI-sum protocol, as illustrated in Figure 18, realizes the $\mathcal{F}_{\mathsf{PSISum}}$ functionality in Figure 11 against a semi-honest adversary.*

Input of $\mathcal{S}$: $X := \{x_1, \ldots, x_{n_s}\}$ and $\tilde{X} := \{\tilde{x}_1, \ldots, \tilde{x}_{n_s}\}$
Input of $\mathcal{R}$: $Y := \{y_1, \ldots, y_{n_r}\}$
Common: $m := 1.27n_r$, and $k := 3$ hash functions $h_j : \{0,1\}^* \to [m]$, $\mathsf{H} : \{0,1\}^* \times \{0,1\}^\kappa \to \{0,1\}^\kappa$.
Protocol:

1) Steps 1 through 7 are consistent with those of the circuit PSI protocol (Figure 16).
2) For each $i \in [m]$, $\mathcal{S}$ sends $r_i \| \tau_i$, and $\mathcal{R}$ sends $r'_i \| \tau'_i$ to $\mathcal{F}_{\mathsf{2PC}}$. $\mathcal{F}_{\mathsf{2PC}}$ computes a circuit that, for each $i \in [m]$, sets $q_i := 1$ if $r_i = r'_i$, and $q_i := 0$ otherwise, and then computes $\mathsf{sum} := \sum_{i=1}^{m} q_i(\tau'_i \oplus \tau_i)$, outputting the result to $\mathcal{R}$.

Fig. 18: Our PSI-sum protocol.

Proof. The proof is trivial and follows similarly to the proof of Theorem 4. □

Input of $\mathcal{S}$: $X := \{x_1, \ldots, x_{n_s}\}$ and $\tilde{X} := \{\tilde{x}_1, \ldots, \tilde{x}_{n_s}\}$
Input of $\mathcal{R}$: $Y := \{y_1, \ldots, y_{n_r}\}$ and $\tilde{Y} := \{\tilde{y}_1, \ldots, \tilde{y}_{n_r}\}$
Common: $m := 1.27n_r$, and $k := 3$ hash functions $h_j : \{0,1\}^* \to [m]$, $\mathsf{H} : \{0,1\}^* \times \{0,1\}^\kappa \to \{0,1\}^\kappa$.
Protocol:

1) Steps 1 through 7 are consistent with those of the circuit PSI protocol (Figure 16).
2) For each $i \in [m]$, $\mathcal{S}$ sends $r_i \| \tau_i$, and $\mathcal{R}$ sends $(r'_i \| \tau'_i, t_{\tilde{y}})$ to $\mathcal{F}_{\mathsf{2PC}}$. $\mathcal{F}_{\mathsf{2PC}}$ computes a circuit that, for each $i \in [m]$, sets $q_i := 1$ if $r_i = r'_i$, and $q_i := 0$ otherwise, and then computes $\mathsf{sum} := \sum_{i=1}^{m} q_i(\tau'_i \oplus \tau_i) \cdot t_{\tilde{y}}$, outputting the result to $\mathcal{R}$.

Fig. 19: Our PJC protocol.

PJC. PJC generalizes PSI-sum by allowing both parties to associate payloads with their inputs. When all payloads on the receiver's side are fixed to 1, the functionality reduces to that of PSI-sum. The PJC protocol is described formally in Figure 19. Therefore, PJC can be realized by slightly modifying the PSI-sum circuit to compute $\mathsf{sum} := \sum_{i=1}^{m} q_i(\tau'_i \oplus \tau_i) \cdot t_{\tilde{y}}$.

Theorem 7. *Our PJC protocol, as illustrated in Figure 19, realizes the $\mathcal{F}_{\mathsf{PJC}}$ functionality in Figure 12 against a semi-honest adversary.*

Proof. The proof is trivial and follows similarly to the proof of Theorem 4. □

VII. Implementation and Evaluation

In this section, our work is evaluated and comprehensively compared with existing works.

A. Experimental Setup

All experiments were conducted on a personal laptop running Ubuntu 20.04, equipped with 16GB of RAM and an 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz CPU. Our implementation is written in C++, where all random oracles are instantiated through batched SHA-256 evaluations accelerated by SHA2-NI intrinsics. The protocol performance can be further improved by leveraging AES-NI, and the corresponding results are presented in Appendix A. In the WAN setting, the protocols are evaluated under 100 Mbps and 10 Mbps bandwidth constraints, with throttling implemented using the Linux `tc` command. We select two state-of-the-art OKVS constructions for our experiments: RR [16], which offers better encoding and decoding efficiency, and BPSY [17], which features lower redundancy. To ensure a fair comparison, protocols that also rely on OKVS are evaluated using the same OKVS. The base random OTs are instantiated using SimplestOT [57]. For existing protocols that require OTE, we adopt [28], while for those that require VOLE, we use [29]. For our circuit PSI and other PSI variants, circuit evaluation is performed using the SPU library [58].

Baseline. The baselines and implementation sources for PSI and its variants used in our experiments are presented below.

- **PSI**. KKRT [12] and the VOLE-based PSI [15]–[17] are selected as baselines for evaluating our work in the standard PSI setting, as they represent the most efficient existing approaches in the OTE-based and VOLE-based PSI categories, respectively. The implementations of KKRT and the VOLE-based PSI are obtained from [59] and [60], respectively. When a comparison using BPSY is required, the OKVS component in [60] is replaced with the implementation from [61].
- **Circuit PSI**. For circuit PSI, four recent and representative existing works are selected: PSTY [22], CGS [23], PEPSI [24], and the VOLE-based circuit PSI [15]–[17]. The implementations of PSTY, CGS, and PEPSI can be found in [62], [63], and [64], respectively. The implementation of the VOLE-based circuit PSI protocol is available in [60]. Similar to the PSI case, when comparing with BPSY as the OKVS, the OKVS component in the circuit PSI implementation of [60] is replaced with that from [61].
- **PSI-cardinality**. For PSI-cardinality, comparisons are made with PSTY [22], GMR$^+$ [31], PEPSI [24], and CZZ$^+$ [21]. There is no official implementation of GMR$^+$ [31], but an implementation can be derived by modifying that of PSTY [62]. The implementations of PSTY and PEPSI are as noted in the circuit PSI part, and the source code for CZZ$^+$ is available at [65].
- **PSI-sum**. For PSI-sum, we compare against PSTY [22], PEPSI [24], and CZZ$^+$ [21], all of which have official implementations that have been provided above.
- **PJC**. For PJC, we compare against IKN$^+$ [5], using its official implementation [66]. We were unable to compare with [51], [52] due to the lack of publicly available implementations, and unlike GMR$^+$ [31], they cannot be reproduced by simple code modifications.
- **Unbalanced setting**. Beyond these functionalities of PSI, we further evaluate the performance of our proposed PSI in the unbalanced setting. We use the state-of-the-art APSI [32] and the recently proposed CM [33] as baselines for comparison. Their official implementations are available at [67] and [68], respectively.

It is worth noting that, similar to our work, a circuit PSI

TABLE III: Runtime and communication comparison between our work and existing PSI protocols under various network settings. If the existing work is optimal, it is highlighted in green; if the proposed work is optimal, it is highlighted in red. A "-" in the runtime column denotes that the implementation of the protocol encountered runtime errors under the given parameter setting.

PSI	Security	OKVS	Running Time (s)									Communication (MB)		
			LAN			WAN (100 Mbps)			WAN (10 Mbps)					
			2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}
KKRT [12]	Semi-Honest	/	0.14	2.6	48	0.7	11.5	197.4	5.5	91.1	1511.6	6.89	114.2	1893.4
VOLE-based PSI [15], [16]	Semi-Honest	RR [16]	0.04	1.03	22.1	0.2	3.7	66.5	1.8	26	464.5	2.25	32.44	490.9
VOLE-based PSI [15], [16]	Malicious	RR [16]	0.05	1.09	23.2	0.3	4.1	75.6	2.8	31.5	501.4	3.34	37.1	571.9
VOLE-based PSI [15], [16]	Semi-Honest	BPSY [17]	0.05	1.13	24.9	0.3	3.3	60.3	2.2	22.4	394.8	2.69	27.56	440.78
VOLE-based PSI [15], [16]	Malicious	BPSY [17]	0.06	1.164	25.2	0.3	3.8	67.5	2.4	26.8	454	3.03	33.56	520.78
Ours	Semi-Honest	RR [16]	0.02	0.74	14.4	0.2	3.3	56.1	1.6	25.8	447.7	1.9	30.02	490.8
Ours	Malicious	RR [16]	0.03	0.76	14.6	0.2	3.8	62.8	1.9	29.1	483.7	2.34	36.02	570.8
Ours	Semi-Honest	BPSY [17]	0.03	0.88	15.1	0.2	3.1	49.0	1.4	21.9	381.4	1.59	26.48	439.68
Ours	Malicious	BPSY [17]	0.04	0.90	15.9	0.2	3.6	56.7	1.7	26.2	449.4	2.03	32.48	519.68

TABLE IV: Runtime and communication comparison between our work and existing circuit PSI protocols under various network settings. If the existing work is optimal, it is highlighted in green; if the proposed work is optimal, it is highlighted in red. A "-" is shown in the OKVS column to indicate that the corresponding protocols do not utilize OKVS. A "-" in the runtime column denotes that the implementation of the protocol encountered runtime errors under the given parameter setting.

Circuit PSI	OKVS	Running Time (s)									Communication (MB)		
		LAN			WAN (100 Mbps)			WAN (10 Mbps)					
		2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}
PSTY [22]	/	3.2	50.4	796.3	15.1	247.3	3702.2	120.9	2006.1	-	148.5	2376	38016
CGS [23]	/	2.3	-	-	18.4	-	-	172.7	-	-	208.6	-	-
PEPSI [24]	/	7	125.3	-	13.1	190.5	-	70.5	741.3	-	75.7	787.7	-
VOLE-based CPSI [15], [16]	RR [16]	1.2	16.8	-	1.8	25.6	-	7.1	106.7	-	7.27	115.1	-
VOLE-based CPSI [15], [16]	BPSY [17]	1.5	18.9	-	2	27.2	-	6.7	103.9	-	6.66	105.53	-
Ours	RR [16]	0.39	4.62	94.1	0.95	13.1	249.8	5.9	88.4	1603.7	6.84	104.6	1884.2
Ours	BPSY [17]	0.42	5.9	121.2	0.7	11.5	211.2	3.9	62.2	1024.9	4.4	70.37	1126

TABLE V: Runtime and communication comparison between our work and existing PSI-cardinality protocols under various network settings. If the existing work is optimal, it is highlighted in green; if the proposed work is optimal, it is highlighted in red. A "–" is shown in the OKVS column to indicate that the corresponding protocols do not utilize OKVS. A "–" in the runtime column denotes that the implementation of the protocol encountered runtime errors under the given parameter setting.

PSI-cardinality	OKVS	Running Time (s)									Communication (MB)		
		LAN			WAN (100 Mbps)			WAN (10 Mbps)					
		2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}
PSTY [22]	/	3.6	54.7	821.6	15.7	254.9	3498.6	125.9	2103.4	-	151	2416	38656
GMR$^+$ [31]	/	10.4	154.2	-	15.1	237.1	-	55.7	1010.3	-	55.49	1030	-
CZZ$^+$ [21]	/	5.4	67.4	1159.4	5.7	73.2	1247.3	8.9	127.1	2076.9	4.46	71.3	1140
PEPSI [24]	/	6.6	125.6	-	12.5	188.7	-	65.4	727.7	-	74.6	769.9	-
Ours	RR [16]	0.27	3.4	95.1	1.4	20.1	379.1	11.0	173.4	2989.7	13.34	208.6	3548.2
Ours	BPSY [17]	0.31	5.8	125.3	1.3	20.1	349.5	9.1	145.3	2358.3	10.9	174.37	2790

can be adapted to support other PSI variants such as PSI-cardinality, PSI-sum, and PJC by replacing the corresponding downstream circuit. However, we do not include all circuit PSI baselines in the evaluations of these variants. This is because existing implementations do not support such functionalities, and modifying their downstream circuits, especially in general-purpose two-party computation, would require significant engineering effort. Therefore, only the functionalities that are already available in the original implementations are considered for comparison.

B. Evaluation and Experimental Results

We now compare our work with existing protocols across different functionalities. Evaluations are performed under LAN and WAN settings with bandwidths of 100 Mbps and 10 Mbps, respectively, to demonstrate the practical performance under high, typical, and low bandwidth conditions.

PSI. As shown in Table III, the proposed PSI protocol outperforms both KKRT [12] and the VOLE-based PSI [15]–[17] in terms of runtime and communication cost. In the LAN setting, the proposed protocol is approximately $1.5\times$ faster than the VOLE-based PSI under the same OKVS and security level, and more than $3\times$ faster than KKRT. For example, when the input size is 2^{24} and RR [16] is used as the OKVS, our PSI protocol completes in 14.4 seconds under the semi-honest model, while VOLE-based PSI with RR takes 22.1 seconds, and KKRT takes 48 seconds. In the WAN setting, as the bandwidth decreases, the performance advantage of our protocol over KKRT remains relatively stable, while the advantage over the VOLE-based PSI protocol diminishes. This is because, in extremely low-bandwidth environments, the performance of even highly optimized protocols becomes primarily determined by communication. Due to Theorem 3,

TABLE VI: Runtime and communication comparison between our work and existing PSI-sum protocols under various network settings. If the existing work is optimal, it is highlighted in green; if the proposed work is optimal, it is highlighted in red. A "-" is shown in the OKVS column to indicate that the corresponding protocols do not utilize OKVS. A "-" in the runtime column denotes that the implementation of the protocol encountered runtime errors under the given parameter setting.

PSI-sum	OKVS	Running Time (s)									Communication (MB)		
		LAN			WAN (100 Mbps)			WAN (10 Mbps)					
		2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}
PSTY [22]	/	3.4	52.6	813.4	15.4	255	3780.1	130.2	1995	-	151	2416	38656
CZZ$^+$ [21]	/	5.1	59.5	986.3	5.6	66.8	1121.8	9.8	134.9	2314.5	5.75	95.3	1642
PEPSI [24]	/	6.4	122.6	-	12.6	182.9	-	64.8	715.6	-	74.6	769.9	-
Ours	RR [16]	0.31	3.5	99.3	1.5	22.2	416.8	12.2	189.6	3247.3	14.84	232.6	3932.2
Ours	BPSY [17]	0.34	5.8	128.9	1.4	21.7	383.9	10.4	165.5	2570.1	12.4	198.37	3174

TABLE VII: Runtime and communication comparison between our work and existing PJC protocols under various network settings. If the existing work is optimal, it is highlighted in green; if the proposed work is optimal, it is highlighted in red. A "-" is shown in the OKVS column to indicate that the corresponding protocols do not utilize OKVS. A "-" in the runtime column denotes that the implementation of the protocol encountered runtime errors under the given parameter setting.

PJC	OKVS	Running Time (s)									Communication (MB)		
		LAN			WAN (100 Mbps)			WAN (10 Mbps)					
		2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}
IKN$^+$ [5]	/	221.9	2697.4	-	225.3	2752.9	-	255.4	3231.8	-	43.2	690.8	-
Ours	RR [16]	0.31	3.69	94.5	1.6	23.6	429.6	13.1	203.1	3445.9	15.84	248.6	4188.2
Ours	BPSY [17]	0.33	6.4	131.3	1.4	22.8	405.7	11.1	180.2	2875.4	13.4	214.37	3430

the proposed PSI protocol will not be slower than the VOLE-based PSI even as the bandwidth continues to decrease. Furthermore, regarding the choice of OKVS, RR [16] shows clear advantages in high-bandwidth environments. However, in WAN settings, BPSY [17] becomes preferable due to its significantly lower redundancy rate of approximately $\epsilon := 1.03$, compared to around 1.3 for RR.

Circuit PSI. Table IV shows that the proposed protocol outperforms existing circuit PSI protocols in both runtime and communication efficiency. Under the LAN setting, the proposed protocol is $3.6\times$ faster than the VOLE-based circuit PSI [15], [16] and approximately $27\times$ faster than PEPSI [24]. For example, when the input size is 2^{20}, our circuit PSI requires only 4.62 seconds when the OKVS is instantiated with RR [16], whereas the VOLE-based circuit PSI and PEPSI require 16.8 seconds and 125.3 seconds, respectively. In terms of communication, it reduces the cost by a factor of 1.5 compared to the former, and by more than $10\times$ compared to the latter. Compared to the classical PSTY [22], the proposed circuit PSI demonstrates remarkable improvements, achieving more than an $8\times$ reduction in runtime and over a $30\times$ reduction in communication. As expected, and similar to the standard PSI setting, our circuit PSI shows better performance in high-bandwidth environments (e.g., LAN) when using RR [16] as the OKVS, while BPSY [17] yields better performance in WAN settings. Note an issue with BPSY that should be taken into account in practice. That is, the encoding parameters remain unclear when the input size is small (i.e., $n_r < 2^{10}$), as also noted in [21]. Therefore, BPSY may be unsuitable for small input size scenarios. The same applies to the other PSI variants and will be omitted from further discussion.

PSI-cardinality. In Table V, the proposed PSI-cardinality protocol is compared with existing protocols [21], [22], [24], [31]. Compared to previous protocols, the proposed PSI-sum protocol achieves lower computational cost, but incurs higher communication overhead than CZZ$^+$ [21]. This is because our protocol requires cuckoo hashing for element alignment and incurs additional communication due to the redundancy of the OKVS. In contrast, CZZ$^+$ primarily relies on public-key operations, resulting in lower communication cost by trading computation for communication. Therefore, our protocol exhibits a significant advantage in computational efficiency, achieving a $12.4\times$ speedup over CZZ$^+$ under the LAN setting. As a result, it is only about 12% slower than CZZ$^+$ even at a bandwidth of 10 Mbps. Specifically, when $n = 2^{24}$, our protocol completes in 95.1 seconds under the LAN setting, while CZZ$^+$ requires 1159.4 seconds. Under a 10 Mbps bandwidth, our protocol takes approximately 2358.3 seconds, compared to 2076.9 seconds for CZZ$^+$, showing that the performance gap is not significant in such low-bandwidth conditions. As for the other three baselines, PSTY [22], GMR$^+$ [31], and PEPSI [24], the proposed protocol demonstrates substantial advantages in both computational and communication costs.

PSI-sum. The conclusions for PSI-sum are similar to those for PSI-cardinality, as shown in Table VI. While the proposed PSI-sum does not outperform CZZ$^+$ [21] in terms of communication, it achieves a significant advantage in computational cost. As in the previous case, when the bandwidth becomes very limited, the runtime of our protocol may become slower than that of CZZ$^+$. However, because it is approximately $10\times$ faster under the LAN setting in terms of computation, the performance gap remains small even at 10 Mbps. When $n = 2^{24}$, under the LAN setting, our protocol completes in 99.3 seconds, while CZZ$^+$ takes 986.3 seconds. Under the 10 Mbps setting, our protocol takes approximately 2570.1 seconds, compared to around 2314.5 seconds for CZZ$^+$, resulting in only a 10% performance difference. For the other two

TABLE VIII: Runtime and communication comparison between our malicious PSI using RR [16] as the OKVS and existing unbalanced PSI protocols [32], [33] under various network settings. If the existing work is optimal, it is highlighted in green; if the proposed work is optimal, it is highlighted in red. A "-" denotes that the implementation of the protocol encountered runtime errors under the given parameter setting. Note that for [32], [33], we omit their offline time since it is executed only once, whereas our PSI does not involve an offline–online phase and thus reports the total end-to-end runtime.

n_r	n_s	APSI [32]				CM [33]				Ours			
		Running Time (s)			Com. (MB)	Running Time (s)			Com. (MB)	Running Time (s)			Com. (MB)
		LAN	WAN (100 Mbps)	WAN (10 Mbps)		LAN	WAN (100 Mbps)	WAN (10 Mbps)		LAN	WAN (100 Mbps)	WAN (10 Mbps)	
2^8	2^{22}	1.67	1.86	3.57	2.38	1.06	1.74	7.91	8.56	0.29	5.41	51.50	64.01
2^{10}		1.63	1.82	3.64	2.43	4.17	6.91	31.57	34.25	0.33	5.45	51.56	64.04
2^{12}		2.66	3.13	7.37	5.89	16.91	27.86	126.43	136.9	0.34	5.47	51.97	64.15
2^{14}		9.64	10.97	22.95	16.64	71.04	114.87	509.36	547.9	0.36	5.51	52.89	64.37
2^{16}		17.66	23.94	71.73	63.32	-	-	-	-	0.49	5.98	55.44	65.35
2^8	2^{24}	5.81	6.12	8.95	3.92	1.07	1.75	7.92	8.56	1.27	21.75	206.09	256.02
2^{10}		5.80	6.15	8.97	3.96	4.21	6.95	31.61	34.25	1.32	21.80	207.15	256.04
2^{12}		7.41	8.02	13.48	7.59	16.91	27.87	126.51	137.0	1.37	21.86	209.37	256.12
2^{14}		12.36	30.78	43.56	17.75	72.21	116.04	510.53	547.9	1.42	21.93	211.52	256.37
2^{16}		38.73	43.85	90.17	63.02	-	-	-	-	1.50	23.28	217.76	257.35
2^{18}		111.6	138.3	328.8	251.9	-	-	-	-	1.66	24.15	221.1	261.13
2^8	2^{26}	-	-	-	-	1.07	1.75	7.92	8.56	4.95	86.87	825.17	1024.02
2^{10}		-	-	-	-	4.32	7.06	31.72	34.25	5.42	88.34	827.65	1024.04
2^{12}		-	-	-	-	17.32	28.28	126.92	137.0	5.51	89.44	829.80	1024.11
2^{14}		-	-	-	-	76.09	119.92	514.41	547.9	5.62	90.57	831.12	1024.37
2^{16}		-	-	-	-	-	-	-	-	5.79	92.25	837.41	1025.35

baselines [22], [24], the proposed protocol shows substantial improvements in both computation and communication.

PJC. For PJC, as shown in Table VII, the proposed protocol achieves comprehensive improvements over IKN$^+$ [5] in both computation and communication costs. For example, when the input size is $n = 2^{20}$, our protocol completes in just 3.69 seconds, while IKN$^+$ [5] requires 2697.4 seconds, resulting in an approximate 731× speedup. Moreover, the communication cost of IKN$^+$ is more than 3× that of our protocol. As the bandwidth decreases, the performance gap narrows. However, even at 10 Mbps, our protocol still achieves an 18× performance improvement due to the substantial gap in computational cost.

Unbalanced setting. The performance evaluation and comparison results for the unbalanced PSI are summarized in Table VIII, where the sender set is significantly larger than that of the receiver ($n_s \gg n_r$). We choose RR [16] as the OKVS since the parameter settings of BPSY [17] remain unclear when $n_r < 2^{10}$. Our comparison method fixes a large n_s and gradually increases n_r to observe the performance of our PSI and that of [32], [33] under different network environments. It can be observed that our PSI achieves a significant advantage over existing state-of-the-art unbalanced PSI protocols [32], [33] in the LAN environment, achieving improvements of up to one or two orders of magnitude. When $n_s = 2^{22}$ and $n_r = 2^{14}$, our PSI requires only 0.36 seconds, whereas APSI and CM take 9.64 seconds and 71.04 seconds, respectively. APSI [32] and CM [33] outperform our PSI mainly in communication. Specifically, APSI achieves the lowest overall communication cost, while CM [33] performs better when n_r is very small (e.g., $n_r < 2^8$), under which it generally attains the highest efficiency across most performance metrics. However, as n_r increases, our PSI gradually gains an advantage under the WAN setting. This is because when $n_s \gg n_r$, the communication cost of our PSI grows very slowly with n_r, whereas that of APSI and CM increases linearly. For example, once n_r reaches a moderately large value (e.g., $n_r = 2^{14}$), our PSI remains the most efficient in the 100 Mbps setting. We observe that when $n_s = 2^{22}$ and $n_r \geq 2^{16}$, our PSI becomes the most efficient under the 10 Mbps setting, and when $n_s = 2^{24}$, the threshold shifts to $n_r = 2^{18}$. Hence, we conclude that when the ratio between n_s and n_r is less than 64, our PSI is likely to be the most efficient even under low-bandwidth conditions such as 10 Mbps.

VIII. Conclusion

This work proposes a new PSI paradigm that supports both the semi-honest and malicious security models. It achieves strictly lower communication than the state-of-the-art paradigm. The proposed PSI protocol is further extended to circuit PSI, and various PSI variants are derived by adapting the downstream circuits. Comprehensive evaluations under different network conditions demonstrate that the proposed protocol offers significant advantages over existing works. Our PSI protocol enables secure data circulation while preserving privacy, allowing all parties to realize the value of their data.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (No. 2023YFB3106500), the National Natural Science Foundation of China (No. 62441227), and the Program of China Scholarship Council (No. 202506230132). We thank all the anonymous reviewers and artifact evaluation committee members for their time, effort, and valuable feedback on our work. We thank Dr. Jingwei Hu from Nanyang Technological University for his numerous discussions with us during the revision phase.

Ethics Considerations

This research does not involve human subjects or user data. The proposed protocols are cryptographic in nature and are designed to improve the efficiency of Private Set Intersection (PSI) operations. No security vulnerabilities were discovered or disclosed during the course of this work, and no adversarial behavior was tested against live infrastructures. We acknowledge that PSI protocols may potentially be applicable

in contexts involving sensitive information, such as healthcare, contact tracing, targeted advertising, and collaborative analytics. However, this work focuses solely on the design and evaluation of efficient cryptographic protocols under well-established security models. We do not evaluate or promote any specific application involving real users, personal data, or privacy-sensitive deployments. Therefore, we believe this work does not raise any direct ethical concerns. Nonetheless, we encourage future adopters to carefully consider context-specific ethical, legal, and regulatory implications when deploying our PSI protocols in practice.

REFERENCES

[1] https://security.googleblog.com/2019/06/helping-organizations-do-more-without-collecting-more-data.html.

[2] https://engineering.fb.com/2020/07/10/open-source/private-matching/.

[3] https://signal.org/blog/contact-discovery/.

[4] https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf.

[5] M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, S. Saxena, K. Seth, M. Raykova, D. Shanahan, and M. Yung, "On deploying secure computing: Private intersection-sum-with-cardinality," in *EuroS&P*. IEEE, 2020, pp. 370–389.

[6] B. Pinkas, T. Schneider, and M. Zohner, "Scalable private set intersection based on OT extension," *ACM Trans. Priv. Secur.*, vol. 21, no. 2, pp. 7:1–7:35, 2018.

[7] Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas, "Private set intersection for unequal set sizes with mobile applications," *Proc. Priv. Enhancing Technol.*, vol. 2017, no. 4, pp. 177–197, 2017.

[8] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert, "Mobile private contact discovery at scale," in *USENIX Security Symposium*. USENIX Association, 2019, pp. 1447–1464.

[9] L. Hetz, T. Schneider, and C. Weinert, "Scaling mobile private contact discovery to billions of users," in *ESORICS (1)*, ser. Lecture Notes in Computer Science, vol. 14344. Springer, 2023, pp. 455–476.

[10] B. Pinkas, T. Schneider, and M. Zohner, "Faster private set intersection based on OT extension," in *USENIX Security Symposium*. USENIX Association, 2014, pp. 797–812.

[11] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private set intersection using permutation-based hashing," in *USENIX Security Symposium*. USENIX Association, 2015, pp. 515–530.

[12] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious PRF with applications to private set intersection," in *CCS*. ACM, 2016, pp. 818–829.

[13] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "Spot-light: Lightweight private set intersection from sparse OT extension," in *CRYPTO (3)*, ser. Lecture Notes in Computer Science, vol. 11694. Springer, 2019, pp. 401–431.

[14] M. Chase and P. Miao, "Private set intersection in the internet setting from lightweight oblivious PRF," in *CRYPTO (3)*, ser. Lecture Notes in Computer Science, vol. 12172. Springer, 2020, pp. 34–63.

[15] P. Rindal and P. Schoppmann, "VOLE-PSI: fast OPRF and circuit-psi from vector-ole," in *EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 12697. Springer, 2021, pp. 901–930.

[16] S. Raghuraman and P. Rindal, "Blazing fast PSI from improved OKVS and subfield VOLE," in *CCS*. ACM, 2022, pp. 2505–2517.

[17] A. Bienstock, S. Patel, J. Y. Seo, and K. Yeo, "Near-optimal oblivious key-value stores for efficient psi, PSU and volume-hiding multi-maps," in *USENIX Security Symposium*. USENIX Association, 2023, pp. 301–318.

[18] K. Han, S. Kim, B. Lee, and Y. Son, "Revisiting okvs-based OPRF and PSI: cryptanalysis and better construction," in *ASIACRYPT (8)*, ser. Lecture Notes in Computer Science, vol. 15491. Springer, 2024, pp. 266–296.

[19] G. Ling, P. Tang, F. Tang, S. Sun, S. Ji, and W. Qiu, "Ultra-fast private set intersection from efficient oblivious key-value stores," *IEEE Trans. Dependable Secur. Comput.*, vol. 22, no. 5, pp. 4998–5014, 2025.

[20] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "PSI from paxos: Fast, malicious private set intersection," in *EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 12106. Springer, 2020, pp. 739–767.

[21] Y. Chen, M. Zhang, C. Zhang, M. Dong, and W. Liu, "Private set operations from multi-query reverse private membership test," in *Public Key Cryptography (3)*, ser. Lecture Notes in Computer Science, vol. 14603. Springer, 2024, pp. 387–416.

[22] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai, "Efficient circuit-based PSI with linear communication," in *EUROCRYPT (3)*, ser. Lecture Notes in Computer Science, vol. 11478. Springer, 2019, pp. 122–153.

[23] N. Chandran, D. Gupta, and A. Shah, "Circuit-psi with linear complexity via relaxed batch OPPRF," *Proc. Priv. Enhancing Technol.*, vol. 2022, no. 1, pp. 353–372, 2022.

[24] R. A. Mahdavi, N. Lukas, F. Ebrahimianghazani, T. Humphries, B. Kacsmar, J. A. Premkumar, X. Li, S. Oya, E. Amjadian, and F. Kerschbaum, "PEPSI: practically efficient private set intersection in the unbalanced setting," in *USENIX Security Symposium*. USENIX Association, 2024.

[25] Y. Yang, X. Liang, X. Song, Y. Dong, L. Huang, H. Ren, C. Dong, and J. Zhou, "Maliciously secure circuit private set intersection via spdz-compatible oblivious PRF," *Proc. Priv. Enhancing Technol.*, vol. 2025, no. 2, pp. 680–696, 2025.

[26] F. Kerschbaum, E. Blass, and R. A. Mahdavi, "Faster secure comparisons with offline phase for efficient private set intersection," in *NDSS*. The Internet Society, 2023.

[27] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl, "Efficient two-round OT extension and silent non-interactive secure computation," in *CCS*. ACM, 2019, pp. 291–308.

[28] L. Roy, "Softspokenot: Quieter OT extension from small-field silent VOLE in the minicrypt model," in *CRYPTO (1)*, ser. Lecture Notes in Computer Science, vol. 13507. Springer, 2022, pp. 657–687.

[29] S. Raghuraman, P. Rindal, and T. Tanguy, "Expand-convolute codes for pseudorandom correlation generators from LPN," in *CRYPTO (4)*, ser. Lecture Notes in Computer Science, vol. 14084. Springer, 2023, pp. 602–632.

[30] Z. Li, C. Xing, Y. Yao, and C. Yuan, "Efficient pseudorandom correlation generators for any finite field," in *EUROCRYPT (5)*, ser. Lecture Notes in Computer Science, vol. 15605. Springer, 2025, pp. 145–175.

[31] G. Garimella, P. Mohassel, M. Rosulek, S. Sadeghian, and J. Singh, "Private set operations from oblivious switching," in *Public Key Cryptography (2)*, ser. Lecture Notes in Computer Science, vol. 12711. Springer, 2021, pp. 591–617.

[32] K. Cong, R. C. Moreno, M. B. da Gama, W. Dai, I. Iliashenko, K. Laine, and M. Rosenberg, "Labeled PSI from homomorphic encryption with reduced computation and communication," in *CCS*. ACM, 2021, pp. 1135–1150.

[33] E. Chielle and M. Maniatakos, "Recurrent private set intersection for unbalanced databases with cuckoo hashing and leveled FHE," in *NDSS*. The Internet Society, 2025.

[34] G. Couteau, P. Rindal, and S. Raghuraman, "Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes," in *CRYPTO (3)*, ser. Lecture Notes in Computer Science, vol. 12827. Springer, 2021, pp. 502–534.

[35] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, vol. 3027. Springer, 2004, pp. 1–19.

[36] C. Meadows, "A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party," in *S&P*. IEEE Computer Society, 1986, pp. 134–137.

[37] B. A. Huberman, M. K. Franklin, and T. Hogg, "Enhancing privacy and trust in electronic communities," in *EC*. ACM, 1999, pp. 78–86.

[38] D. J. Bernstein, "Curve25519: New diffie-hellman speed records," in *Public Key Cryptography*, ser. Lecture Notes in Computer Science, vol. 3958. Springer, 2006, pp. 207–228.

[39] C. Costello and P. Longa, "Four$\mathbb{Q}$: Four-dimensional decompositions on a $\mathbb{Q}$-curve over the mersenne prime," in *ASIACRYPT (1)*, ser. Lecture Notes in Computer Science, vol. 9452. Springer, 2015, pp. 214–235.

[40] M. Rosulek and N. Trieu, "Compact and malicious private set intersection for small sets," in *CCS*. ACM, 2021, pp. 1166–1181.

[41] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *CRYPTO*, ser. Lecture Notes in Computer Science, vol. 2729. Springer, 2003, pp. 145–161.

[42] R. Pagh and F. F. Rodler, "Cuckoo hashing," in *ESA*, ser. Lecture Notes in Computer Science, vol. 2161. Springer, 2001, pp. 121–133.

[43] G. Garimella, B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "Oblivious key-value stores and amplification for private set intersection," in *CRYPTO (2)*, ser. Lecture Notes in Computer Science, vol. 12826. Springer, 2021, pp. 395–425.

[44] P. Schoppmann, A. Gascón, M. Raykova, and B. Pinkas, "Make some ROOM for the zeros: Data sparsity in secure distributed machine learning," in *CCS*. ACM, 2019, pp. 1335–1350.
[45] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *NDSS*. The Internet Society, 2012.
[46] M. Ciampi and C. Orlandi, "Combining private set-intersection with secure two-party computation," in *SCN*, ser. Lecture Notes in Computer Science, vol. 11035. Springer, 2018, pp. 464–482.
[47] E. D. Cristofaro, P. Gasti, and G. Tsudik, "Fast and private computation of cardinality of set intersection and union," in *CANS*, vol. 7712. Springer, 2012, pp. 218–231.
[48] S. K. Debnath and R. Dutta, "Secure and efficient private set intersection cardinality using bloom filter," in *ISC*, ser. Lecture Notes in Computer Science, vol. 9290. Springer, 2015, pp. 209–226.
[49] A. Davidson and C. Cid, "An efficient toolkit for computing private set operations," in *ACISP (2)*, ser. Lecture Notes in Computer Science, vol. 10343. Springer, 2017, pp. 261–278.
[50] M. Ion, B. Kreuter, E. Nergiz, S. Patel, S. Saxena, K. Seth, D. Shanahan, and M. Yung, "Private intersection-sum protocol with applications to attributing aggregate ad conversions," *IACR Cryptol. ePrint Arch.*, p. 738, 2017.
[51] T. Lepoint, S. Patel, M. Raykova, K. Seth, and N. Trieu, "Private join and compute from PIR with default," in *ASIACRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 13091. Springer, 2021, pp. 605–634.
[52] Y. Son and J. Jeong, "PSI with computation or circuit-psi for unbalanced sets from homomorphic encryption," in *AsiaCCS*. ACM, 2023, pp. 342–356.
[53] H. Chen, K. Laine, and P. Rindal, "Fast private set intersection from homomorphic encryption," in *CCS*. ACM, 2017, pp. 1243–1255.
[54] H. Chen, Z. Huang, K. Laine, and P. Rindal, "Labeled PSI from fully homomorphic encryption with malicious security," in *CCS*. ACM, 2018, pp. 1223–1237.
[55] M. O. Rabin, "How to exchange secrets with oblivious transfer," Cryptology ePrint Archive, Paper 2005/187, 2005.
[56] D. Bui and G. Couteau, "Improved private set intersection for sets with small entries," in *Public Key Cryptography (2)*, ser. Lecture Notes in Computer Science, vol. 13941. Springer, 2023, pp. 190–220.
[57] T. Chou and C. Orlandi, "The simplest protocol for oblivious transfer," in *LATINCRYPT*, ser. Lecture Notes in Computer Science, vol. 9230. Springer, 2015, pp. 40–58.
[58] J. Ma, Y. Zheng, J. Feng, D. Zhao, H. Wu, W. Fang, J. Tan, C. Yu, B. Zhang, and L. Wang, "Secretflow-spu: A performant and user-friendly framework for privacy-preserving machine learning," in *USENIX ATC*. USENIX Association, 2023, pp. 17–33.
[59] https://github.com/osu-crypto/BaRK-OPRF.
[60] https://github.com/Visa-Research/volepsi.
[61] https://github.com/google/private-membership/tree/main/research/okvs.
[62] https://github.com/encryptogroup/OPPRF-PSI.
[63] https://github.com/shahakash28/2PC-Circuit-PSI.
[64] https://github.com/RasoulAM/pepsi.
[65] https://github.com/yuchen1024/Kunlun/tree/master/mpc.
[66] https://github.com/google/private-join-and-compute.
[67] https://github.com/microsoft/APSI.
[68] https://github.com/momalab/psi-ndss2025.
[69] B. Kreuter, A. Shelat, and C. Shen, "Billion-gate secure computation with malicious adversaries," in *USENIX Security Symposium*. USENIX Association, 2012, pp. 285–300.
[70] M. Keller, E. Orsini, and P. Scholl, "Actively secure OT extension with optimal overhead," in *CRYPTO (1)*, ser. Lecture Notes in Computer Science, vol. 9215. Springer, 2015, pp. 724–741.
[71] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole - reducing data transfer in garbled circuits using half gates," in *EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 9057. Springer, 2015, pp. 220–250.
[72] M. Keller, E. Orsini, and P. Scholl, "MASCOT: faster malicious arithmetic secure computation with oblivious transfer," in *CCS*. ACM, 2016, pp. 830–842.
[73] M. Chase, E. Ghosh, and O. Poburinnaya, "Secret-shared shuffle," in *ASIACRYPT (3)*, ser. Lecture Notes in Computer Science, vol. 12493. Springer, 2020, pp. 342–372.
[74] C. Guo, J. Katz, X. Wang, and Y. Yu, "Efficient and secure multiparty computation from fixed-key block ciphers," in *SP*. IEEE, 2020, pp. 825–841.

APPENDIX A
FURTHER ACCELERATION VIA AES-NI

In several prior works on MPC [69]–[74], the hash functions modeled as random oracles have been instantiated based on fixed-key AES to achieve higher efficiency. In the early stages, this approach was mostly heuristic and lacked formal security analysis. Guo et al. [74] proposed a systematic approach that uses fixed-key AES to implement OTE and formally proved its security. In simple terms, in [74], a hash function H is implemented as $H(x) = \pi(x) \oplus x$, where $\pi(\cdot)$ can be instantiated using highly optimized AES-NI. This method is also applicable to our PSI. Recall Step 3 in Figures 14 and 15, where this method is well-suited for computing $A(y)$ and $B(y)$, with a_i and b_i as the keys and y as the ciphertext. It is sufficient to take the least significant bit of the output as the output of the oracle H^{b}. Other random oracle implementations can also be directly replaced with AES-NI. In the semi-honest setting of our PSI, the output length of other oracles is typically $\log_2(n_s n_r) + \lambda$, which rarely exceeds $\kappa = 128$. In our experiments, it is at most $24 + 24 + 40 = 88$ bits. In the malicious model, the oracle output length in PSI is fixed to κ. In circuit PSI, the PSI mask length is $\log_2 m + \lambda$, where $m = 1.27 n_r$, which clearly almost never exceeds κ. Therefore, directly adopting the implementation of [74] can replace these oracle instantiations. To handle cases where the desired output length exceeds 128 bits (which is not required in our current paper), one can follow the standard approach of deriving multiple independent 128-bit blocks from fixed-key AES under domain separation (e.g., using a counter as an additional input), concatenating them, and truncating the result to the required length. In fact, the official implementation [60] of VOLE-based PSI [15], [16] has already employed AES-NI for acceleration. Our PSI protocol can further achieve performance gains from this technique. Since replacing the hash function instantiation does not affect communication, we report only the runtime of our PSI in the LAN setting when instantiated with SHA2-NI and AES-NI under the semi-honest and malicious models, respectively.

TABLE IX: Running time comparison of our PSI when instantiating the random oracle with SHA2-NI and AES-NI.

Security	OKVS	Running Time (s) (SHA2-NI)			Running Time (s) (AES-NI)		
		2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}
Semi-Honest	RR [16]	0.02	0.74	14.4	0.019	0.603	13.04
Malicious		0.03	0.76	14.6	0.029	0.655	13.37
Semi-Honest	BPSY [17]	0.03	0.88	15.1	0.025	0.76	13.65
Malicious		0.04	0.90	15.9	0.037	0.792	14.06

As shown in Table IX, using AES-NI can further improve the performance of our PSI by approximately 10%–20%. For example, when the OKVS is RR [16] and the input size is 2^{24}, our PSI originally takes 14.4 and 14.6 seconds under the semi-honest and malicious models, respectively, while using AES-NI reduces the runtime to 13.04 and 13.37 seconds. Although the improvement is not substantial, given that our PSI is already highly efficient, achieving further optimization is not easy. Therefore, it can also serve as an optional optimization to further accelerate the performance of PSI.

APPENDIX A
ARTIFACT APPENDIX

A. Abstract

This artifact appendix accompanies the paper "Faster Than Ever: A New Lightweight Private Set Intersection and Its Variants." It provides the implementation details of the proposed protocol and its extensions. We present a highly efficient Private Set Intersection (PSI) protocol that can be further extended to support circuit PSI, PSI-cardinality, PSI-sum, and Private Join and Compute (PJC). The implementation is written entirely in C++, following the C++17 standard, and can be compiled on any modern Linux environment. This implementation serves as concrete evidence of the efficiency claimed in the paper. In particular, our PSI protocol achieves approximately 50% faster runtime than the most efficient existing PSI schemes. Specifically, when both parties hold sets of size 2^{20}, the protocol completes the intersection in about 0.7 seconds with a total communication cost of around 30 MB.

B. Description & Requirements

1) How to access: https://doi.org/10.5281/zenodo.17699084 or https://github.com/ShallMate/OurPSI

2) Hardware dependencies: **Any modern x86-64 CPU with SHA2 instruction extensions (e.g., Intel SHA-NI) and AVX-512 support should be able to reproduce our results. Our local environment uses an 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80 GHz processor. Any CPU with equivalent or higher performance should be able to reproduce the reported results.** We recommend using a machine with at least 16 GB of RAM, as reproducing experiments with set sizes greater than or equal to 2^{24} requires sufficient memory to complete the computation.

3) Software dependencies: Since our implementation is provided via Docker, you only need to have `Docker` and `git` installed. Note that the versions listed below correspond to our local environment and are not strict requirements for running the artifact.

- Docker: version 27.3.1 (on our local environment)
- Git: version 2.25.1 (on our local environment)

C. Artifact Installation & Configuration

We have containerized our PSI implementation using Docker. Two installation options are available:

1. Build the image locally using the provided `Dockerfile`; or
2. Pull the pre-built image directly from Docker Hub.

We emphasize that both installation methods require an active Internet connection. Note that the second approach downloads the full pre-built image, which is approximately 4.28 GB in size. Therefore, if your network bandwidth is limited, we recommend using the first method (building locally from the `Dockerfile`). However, if you encounter issues while building the image with the provided `Dockerfile`, you may resort to the second method as a last resort.

1. Build using the `Dockerfile`. First, download our prepared GitHub repository to your local machine.

```
git clone https://github.com/ShallMate/OurPSI.git
```

Then, navigate to the project directory and run the following command to build the Docker image:

```
cd OurPSI
docker build -t shallmate/ourpsi:latest .
```

2. Pulling the Pre-built Image from Docker Hub. You can directly obtain our pre-built image from Docker Hub using the `docker pull` command.

```
docker pull shallmate/ourpsi:latest
```

D. Experiment Workflow

This implementation corresponds to our PSI, which can be further extended to a Circuit PSI. Other ideal functionalities such as PSI-sum, PSI-cardinality, and Private Join and Compute (PJC) are directly realized based on the Circuit PSI. Therefore, the experimental workflow consists of two parts:

1. Evaluation of our PSI implementation.
2. Evaluation of our Circuit PSI implementation.

E. Major Claims

From the above experimental workflow, it can also be observed that our major claims primarily focus on the PSI and Circuit PSI.

- (C1): Our PSI implementation completes the intersection computation for input sets of size 2^{24} in approximately 15 seconds, achieving a $1.4\times$ to $1.5\times$ speedup over the most efficient existing VOLE-based PSI protocols. This performance is empirically validated by experiments (E1) and (E2), and the reported runtime corresponds to the LAN running time in Table III of our paper.
- (C2): Our proposed Circuit PSI protocol is more than $3\times$ faster than existing circuit PSI schemes and reduces the communication cost by a factor of 1.5. This corresponds to experiments (E3) and (E4), which are reflected in the last two rows of Table IV in this paper.

F. Evaluation

All experiments should be executed within the directory containing the compiled binaries inside the Docker container. Therefore, before running any experiments, you should start the container and navigate to the directory where the executables are located:

```
docker run --rm -it --platform linux/amd64 \
    shallmate/ourpsi:latest /bin/bash
cd /opt/yacl/bazel-bin/examples/otokvspsi/
```

1) Experiment (E1): [Our PSI uses RR as the OKVS] [within one minute]: RR is an OKVS with a redundancy rate of 1.3, which serves as a key component of our PSI protocol. By executing the corresponding binary, you can obtain the running time and communication cost under both the malicious and semi-honest models. **Given that different CPUs may lead to varying performance, we consider any running time**

below 25 **seconds under both the semi-honest and malicious models to be reasonable. However, the communication costs of 570.802 MB and 490.802 MB are fixed and should remain exact without deviation**.

[How to]: You can directly run our executable, but make sure to choose the correct parameters.

[Preparation]: Make sure that you are currently in the `/opt/yacl/bazel-bin/examples/otokvspsi` directory, where the `ourpsi` executable is located.

[Execution]: Directly execute the following command:

```
./ourpsi 0 0 24 24
```

[Results]: Check the output `Execution time` and `Total communication` to obtain the results.

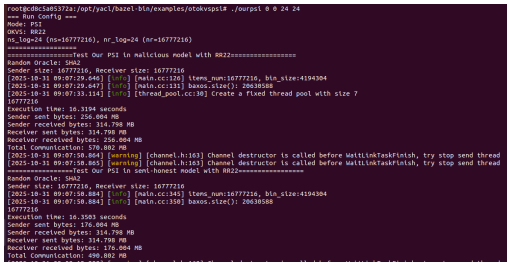

Fig. 20: An example for reproducing PSI using RR as the OKVS.

2) Experiment (E2): [Our PSI uses BPSY as the OKVS] [within one minute]: BPSY is an OKVS with a redundancy rate of 1.03, which is lower than that of RR. However, its computational overhead is slightly higher. By executing the corresponding binary, you can obtain the running time and communication cost under both the malicious and semi-honest models. **Given that different CPUs may lead to varying performance, we consider any running time below** 30 **seconds under both the semi-honest and malicious models to be reasonable. However, the communication costs of 519.685 MB and 439.685 MB are fixed and should remain exact without deviation**.

[How to]: The same as E(1).

[Preparation]: The same as E(1).

[Execution]: Directly execute the following command:

```
./ourpsi 0 1 24 24
```

[Results]: Check the output `Execution time` and `Total communication` to obtain the results.

3) Experiment (E3): [Our Circuit PSI uses RR as the OKVS] [within one minute]: By executing the corresponding binary, you can obtain the running time and communication cost under both the malicious and semi-honest models. Since the execution for input size 2^{24} takes relatively longer, we use an input size of 2^{20} for this experiment. **Given that different CPUs may lead to varying performance, we consider any running time below** 10 **seconds to be reasonable. However, the communication cost of 104.606 MB is fixed and should remain exact without deviation.**

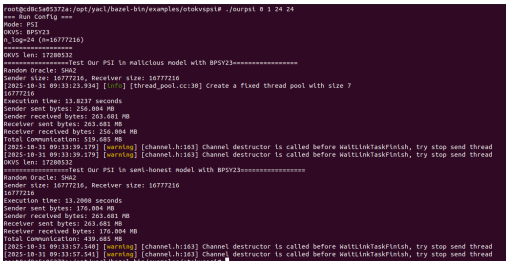

Fig. 21: An example for reproducing PSI using BPSY as the OKVS.

[How to]: The same as E(1). *[Preparation]*: The same as E(1).

[Execution]: Directly execute the following command:

```
./ourpsi 1 0 20 20
```

[Results]: Check the output `Execution time` and `Total communication` to obtain the results.

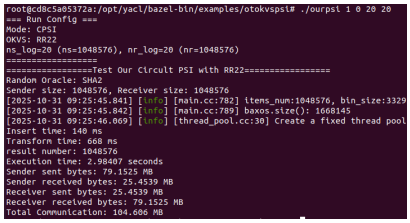

Fig. 22: An example for reproducing circuit PSI using RR as the OKVS.

4) Experiment (E4): [Our Circuit PSI uses BPSY as the OKVS] [within one minute]: By executing the corresponding binary, you can obtain the running time and communication cost under both the malicious and semi-honest models. Since the execution for input size 2^{24} takes relatively longer, we use an input size of 2^{20} for this experiment. **Given that different CPUs may lead to varying performance, we consider any running time below** 30 **seconds to be reasonable. However, the communication cost of 70.3736 MB is fixed and should remain exact without deviation.**

[How to]: The same as E(1). *[Preparation]*: The same as E(1).

[Execution]: Directly execute the following command:

```
./ourpsi 1 1 20 20
```

[Results]: Check the output `Execution time` and `Total communication` to obtain the results.

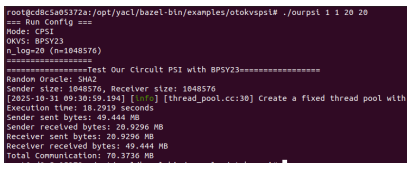

Fig. 23: An example for reproducing circuit PSI using BPSY as the OKVS.