Scilight

Journal of Machine Learning and Information Security
https://www.sciltp.com/journals/jmlis



Article

Check for updates

Flash: Fast, Lightweight Private Set Intersection with Malicious Security Almost for Free

Ze Jiang [1] and Xiangyi Fang [2,*]

[1] College of Computer Science, Chongqing University, Chongqing 400044, China
[2] COMAC Shanghai Aviation Industrial (Group) Co.,Ltd., Shanghai 200232, China
* Correspondence: fangxiangyi@comac.cc

How To Cite: Jiang, Z.; Fang, X. Flash: Fast, Lightweight Private Set Intersection with Malicious Security Almost for Free. *Journal of Machine Learning and Information Security* **2026**, *2*(4), 22. https://doi.org/10.53941/jmlis.2026.100022

Received: 25 June 2026
Revised: 20 October 2026
Accepted: 10 November 2026
Published: 17 November 2026

Abstract: Private set intersection (PSI) allows two parties to learn the intersection of their input sets without revealing any non-matching elements, and underlies applications such as private contact discovery, password-breach detection, and illegal-content detection. Recent state-of-the-art PSI protocols follow a vector oblivious linear evaluation (VOLE) and oblivious key-value store (OKVS) paradigm. While highly efficient, this approach inherits the algorithmic and implementation complexity of OKVS. In this work, we propose a lightweight paradigm based solely on VOLE. Our protocol separates a VOLE phase, which generates all required correlations, from a simple PSI phase that consumes these correlations directly, avoiding any OKVS encoding or decoding. We formally prove its security in both semi-honest and malicious models. Experimental results show that our construction achieves up to 3.7× speedup over state-of-the-art PSI in the LAN setting and up to 2.3× in the WAN setting. The malicious extension adds an average running-time overhead of 15.40% and an average communication overhead of 22.84% across the evaluated settings.

Keywords: private set intersection; multi-party computation

1. Introduction

Private set intersection (PSI) enables two parties, each holding an input set, to learn the intersection of their sets without revealing any additional information about their non-matching elements. Although PSI can in principle be realized using general-purpose secure computation, practical efficiency considerations have motivated protocols tailored to the structure of specific security tasks [1,2]. This primitive has become a standard building block for privacy-preserving data analysis and access control.

PSI has a wide range of practical applications, including private contact discovery, password-breach detection, and illegal-content detection in online services. In private contact discovery, a messaging provider helps a user learn which entries in her address book are already registered on the service, without learning the rest of the address book and without revealing the full user base of the provider [3]. In password-breach detection, a client checks whether its password or password hash appears in a breach corpus held by a security provider, without disclosing the password itself or learning unrelated compromised credentials [4]. In illegal-content detection, a platform tests user-generated content against a blacklist of known abusive or prohibited items (e.g., CSAM hashes or malware signatures) without exposing the entire blacklist or leaking information about benign user data [5]. These scenarios typically involve large sets and repeated executions, so both computation and communication efficiency are critical.

Modern fast PSI protocols largely follow the Oblivious Pseudorandom Function (OPRF)-based paradigm, in which one party holds the PRF key and the other obtains PRF evaluations on its set; the intersection can then be recovered by comparing PRF outputs on both sides. Recent work has shown that OPRFs can be realized very efficiently using Vector Oblivious Linear Evaluations (VOLE) correlations combined with Oblivious Key–Value Stores (OKVS) [6–10]. Concretely, VOLE provides cheap correlated randomness, while OKVS is used to compactly encode and decode large sets. As a result, VOLE+OKVS-based PSI schemes currently achieve state-of-the-art concrete performance.

Copyright: © 2026 by the authors. This is an open access article under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).
Publisher's Note: Scilight stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

While VOLE+OKVS PSI protocols are highly efficient, they inherit the full complexity of general-purpose OKVS constructions. A substantial part of the implementation effort and running time is devoted to building and decoding OKVS instances that support arbitrary key–value mappings, even though PSI only requires a rather structured pattern of correlations. This raises a natural question: in the specific setting of two-party PSI, do we really need a fully-fledged OKVS, or can we simplify the paradigm and rely more directly on fast VOLE correlations?

This work answers the above question affirmatively. We propose a new paradigm for PSI based on random reversed VOLE (rVOLE) [8], which is defined in Section 3. The conventional VOLE-PSI paradigm uses an OKVS to obtain a batched OPRF from VOLE correlations and determines the intersection by matching its outputs [8,9]. Our construction instead associates one rVOLE correlation with each position in the receiver's table and matches hash tags derived from the corresponding affine values. As sketched in Figure 1, our protocol first generates all the necessary correlations, followed by an intersection phase that consumes these correlations directly, without encoding the input set into an OKVS. Building on this paradigm, we obtain a concrete PSI construction.

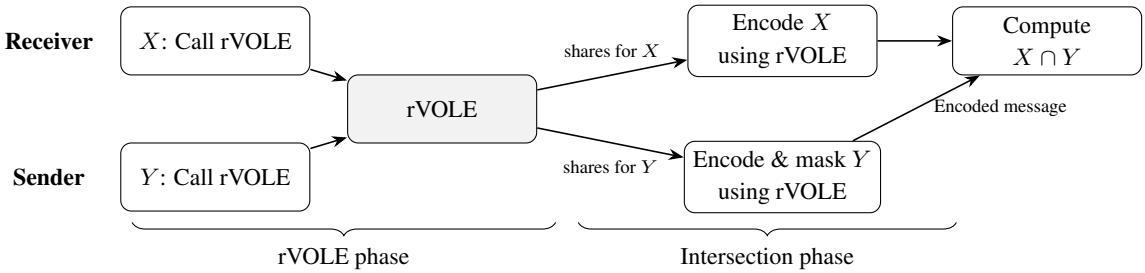


Figure 1. High-level overview of our rVOLE-based PSI protocol (receiver inputs set X and sender inputs set Y).

Our main contributions are as follows:

(1) We introduce a lightweight VOLE-based paradigmfor two-party PSI that replaces the standard VOLE+OKVS approach with a simpler VOLE-only structure tailored to the PSI task, reducing conceptual and implementation complexity.

(2) We give an explicit PSI protocol in this paradigm and extend it from the semi-honest to the malicious setting by simply replacing the underlying rVOLE with a maliciously secure instantiation, while preserving the structure of the PSI phase.

(3) We implement our construction in C++ and evaluate it under various network conditions. The experiments demonstrate that our protocol is competitive with, and often faster than, prior VOLE+OKVS-based PSI schemes, achieving speedups over RR22 [9] of up to about $3.7\times$ on LAN and $2.3\times$ on WAN, while retaining low communication costs.

2. Related Works

Early work on two-party PSI dates back to Meadows [11], who gave one of the first protocols based on the Diffie–Hellman key agreement. Later constructions based on homomorphic encryption (HE), such as [12,13], improved the asymptotic security guarantees but still suffer from relatively high concrete costs due to expensive public-key operations, which limit their practicality for large-scale deployments.

A major shift towards practically efficient PSI came with the OPRF-based paradigm of Hazay and Lindell [14]. This OPRF-based paradigm has since become the dominant framework for fast two-party PSI. With the advent of efficient OT extension, a long line of works [15–21] used OT-based OPRFs together with hash-to-bin techniques to obtain highly optimized PSI protocols with low communication and very fast online phases.

More recently, the introduction of oblivious key–value stores (OKVS) [6,7] and VOLE-based OPRFs has led to another leap in efficiency. PSI schemes following this VOLE+OKVS paradigm, such as [8–10], reduce both computation and bandwidth by encoding large sets compactly and offloading most heavy work to an offline correlation phase. These protocols represent the current state of the art in two-party PSI performance.

In parallel, many variants of PSI with richer functionality have been studied. Unbalanced PSI focuses on settings where one set is much larger than the other and optimizes the work of the smaller party [22,23]. Labeled PSI extends standard PSI so that, in addition to the intersection, the receiver learns the associated values attached to the intersecting elements [24,25]. Multi-party PSI generalizes the functionality to more than two parties and aims at efficiently computing the intersection of many mutually distrustful clients [26,27]. PSI cardinality (PSI-CA) reveals only the size of the intersection without disclosing the elements themselves [28,29].

3. Preliminaries

3.1. Notations

We use λ to denote the statistical security parameter and κ for the computational security parameter. Let $\mathbb{F}$ be a finite field. All algebraic operations are over $\mathbb{F}$ unless stated otherwise. We write ℓ for the output length of hash functions and random oracles, and use "$\|$" to denote the concatenation of bit strings. Vectors over $\mathbb{F}$ are written in bold or with an arrow, e.g., $\vec{R} = (R_1, \ldots, R_m)$, and for $m \in \mathbb{N}$ we use $[m]$ as shorthand for the index set $\{1, \ldots, m\}$. Input sets are denoted by capital letters such as $X, Y \subseteq \mathbb{F}$ with $|X|, |Y| \leq n$ for a public bound n. We write $x \xleftarrow{\$} S$ for sampling x uniformly at random from a finite set S. The symbol $\perp$ denotes a distinguished failure or empty value. Negligible functions in the security parameter are written as $\mathsf{negl}(\cdot)$, and all logarithms are base 2.

3.2. Private Set Intersection

In a two-party PSI protocol, a receiver holding a set X and a sender holding a set Y want to learn the intersection $X \cap Y$ without revealing any additional information about their inputs. In this work, we consider the standard asymmetric variant where only the receiver learns the intersection and the sender learns nothing beyond what can be inferred from its own input and the fact that the protocol completed successfully. The ideal functionality $\mathcal{F}_{\mathsf{PSI}}$ is shown in Figure 2, which is borrowed from [9]. $\mathcal{F}_{\mathsf{PSI}}$ allows public upper bounds n_X, n_Y on the receiver and sender set sizes, respectively, with n_X applying to an honest receiver. A malicious receiver's effective input is bounded by $n'_X \geq n_X$.

Ideal Functionality $\mathcal{F}_{\mathsf{PSI}}$ (Private Set Intersection)

Parameters. There are two parties, a sender with set $Y \subseteq \mathbb{F}$ and a receiver with set $X \subseteq \mathbb{F}$. Let $n_Y, n_X, n'_X \in \mathbb{Z}_{>0}$ be public parameters with $n_X \leq n'_X$.

Functionality. Upon receiving $(\mathsf{sender}, \mathsf{sid}, Y)$ from the sender and $(\mathsf{receiver}, \mathsf{sid}, X)$ from the receiver, the functionality proceeds as follows:

- If $|Y| > n_Y$, then output $\perp$ to both parties and abort.
- If the receiver is malicious and $|X| > n'_X$, then output $\perp$ to both parties and abort.
- If the receiver is honest and $|X| > n_X$, then output $\perp$ to both parties and abort.
- Otherwise, compute the intersection $I := X \cap Y$ and send I to the receiver. The sender receives no output (or an empty dummy symbol $\perp$).

Figure 2. Ideal functionality $\mathcal{F}_{\mathsf{PSI}}$ for two-party private set intersection.

3.3. Cuckoo Hashing

We make use of a standard Cuckoo hashing scheme, written $\mathsf{CH} = (\mathsf{CH.Setup}, \mathsf{CH.Hash})$, which is parameterized by a field of elements $\mathbb{F}$ and a fixed number b of hash functions.

- $\mathsf{CH.Setup}(\lambda, n, \varepsilon) \to (m, h_1, \ldots, h_b)$: on input a security parameter λ, an upper bound n on the set size, and an overload parameter $\varepsilon > 0$, the setup algorithm returns a table length $m \in \mathbb{N}$ and b hash functions $h_j : \mathbb{F} \to [m]$.
- $\mathsf{CH.Hash}(h_1, \ldots, h_b, D) \to T$ or $\perp$: given the hash functions and a finite set $D \subseteq \mathbb{F}$, the (deterministic) hashing algorithm outputs a table $T = [T[1], \ldots, T[m]]$, where each position $T[i]$ is either an element of D or a distinguished empty symbol $\perp$. If all elements cannot be placed, the algorithm returns $\perp$ and discards the partial table.

We require the following correctness properties from the Cuckoo hashing scheme. First, for every $x \in \mathbb{F}$ the b candidate positions $h_1(x), \ldots, h_b(x)$ are pairwise distinct. Second, each non-empty entry of the table is unique: if $T[i] \neq \perp$ and $T[j] \neq \perp$ with $i \neq j$, then $T[i] \neq T[j]$. Third, for any $n \in \mathbb{N}$ and any set $D \subseteq \mathbb{F}$ of size $|D| \leq n$, it holds with probability at least $1 - \mathsf{negl}(\lambda)$ (over $(m, h_1, \ldots, h_b) \leftarrow \mathsf{CH.Setup}(\lambda, n, \varepsilon)$ and $T \leftarrow \mathsf{CH.Hash}(h_1, \ldots, h_b, D)$) that $T \neq \perp$ and, for every $x \in D$, we have $T[h_j(x)] = x$ for some $j \in \{1, \ldots, b\}$.

Insertion uses an empty candidate position when available; otherwise it displaces an existing entry and reinserts that entry. On failure, fresh independent hash functions are sampled and insertion is repeated until all elements are placed. The table size and number of hashes remain fixed. For per-trial failure probability $p_{\mathsf{CH}} \leq \mathsf{negl}(\lambda)$, the expected number of trials is $1/(1 - p_{\mathsf{CH}})$ and the probability of r consecutive failures is at most p_{CH}^r.

3.4. Vector Oblivious Linear Evaluation

We borrow the ideal functionality $\mathcal{F}_{\mathsf{rVOLE}}$ for *random reversed vector-OLE* from [8] (see Figure 3). In this functionality, there are two parties, a Sender and a Receiver, and neither party provides any input. The functionality samples a field element $\Delta \leftarrow \mathbb{F}$ and two random vectors $\vec{R}, \vec{K} \leftarrow \mathbb{F}^m$, and sets $\vec{M} := \vec{K} + \vec{R}\Delta$. It then delivers the pair $(\Delta, \vec{K})$ to the Sender and the pair $(\vec{R}, \vec{M})$ to the Receiver. Thus, the parties obtain random correlated values of the form

$$\vec{M} = \vec{R}\Delta + \vec{K},$$

without providing any inputs themselves.

Parameters. There are two parties, a Sender and a Receiver. Let $\mathbb{F}$ be a field. Let m denote the size of the output vectors.
Functionality. Upon receiving $(\mathsf{sender}, \mathsf{sid})$ from the Sender and $(\mathsf{receiver}, \mathsf{sid})$ from the Receiver:

- If the Receiver is malicious, wait for it to send $\vec{M}, \vec{R} \in \mathbb{F}^m$. Sample $\Delta \leftarrow \mathbb{F}$ and compute $\vec{K} := \vec{M} - \vec{R}\Delta$.
- If the Sender is malicious, wait for it to send $\vec{K} \in \mathbb{F}^m$ and $\Delta \in \mathbb{F}$. Sample $\vec{R} \leftarrow \mathbb{F}^m$ and compute $\vec{M} := \vec{K} + \vec{R}\Delta$.
- Sample $\vec{R}, \vec{K} \leftarrow \mathbb{F}^m$ and $\Delta \leftarrow \mathbb{F}$ and compute $\vec{M} := \vec{K} + \vec{R}\Delta$.

The functionality sends $(\Delta, \vec{K})$ to the Sender and $(\vec{R}, \vec{M})$ to the Receiver, where $\vec{M} := \vec{R}\Delta + \vec{K}$.

Figure 3. Ideal functionality $\mathcal{F}_{\mathsf{rVOLE}}$.

4. Construction

We consider a two-party PSI setting with a sender and a receiver, where each party holds an input set and the receiver learns the intersection while the sender learns nothing beyond what can be inferred from its own input. We assume authenticated channels and work in the random-oracle model. We aim to keep communication and computation low while achieving semi-honest security, security against malicious receivers, and receiver-input privacy against malicious senders.

4.1. Toy Construction

We first present a toy protocol, where receiver inputs $X = \{x_1, \ldots, x_n\}$ and sender inputs $Y = \{y_1, \ldots, y_n\}$.

(1) Receiver and sender invokes $\mathcal{F}_{\mathsf{rVOLE}}$, receiver gets $\vec{R} = (r_1, \ldots, r_n), \vec{M} = (M_1, \ldots, M_n)$ and sender gets $\vec{K} = (K_1, \ldots, K_n), \Delta$, such that $\vec{M} = \vec{K} + \vec{R}\Delta$.
(2) Receiver computes $E = (e_i)_{i=1}^n$ with $e_i = r_i - x_i$ and sends E to sender. Sender updates $K_i := K_i + e_i\Delta$ for all $i \in [n]$.
(3) Sender sends $(\mathcal{M}_j)_{j=1}^n$, where $\mathcal{M}_j = (L_i(y_j))_{i=1}^n$ and $L_i(y_j) = K_i + y_j\Delta$.
(4) Receiver outputs $I := \{x_i : M_i = L_i(y_j) \text{ for some } j \in [n]\}$.

The above construction is correct whenever $\Delta \neq 0$. If $x_i = y_j$ then $M_i = K_i + x_i\Delta = K_i + y_j\Delta = L_i(y_j)$, so x_i is included in I; if $x_i \neq y_j$ for all j then $M_i \neq L_i(y_j)$ for all j, so $x_i \notin I$. However, the protocol is extremely inefficient. The sender computes and sends n tags for each of the n elements y_j, i.e., a total of n^2 field elements. Both parties also perform $O(n^2)$ field operations, so the total communication and computation cost is quadratic in n.

More importantly, this toy protocol is *insecure* because the values $L_i(y_j) = K_i + y_j\Delta$ preserve too much algebraic structure. From the receiver's point of view, for any fixed index i the map $y \longmapsto L_i(y) = K_i + y\Delta$ is an affine function of the sender's input y over $\mathbb{F}$. Since the receiver learns *all* values $L_i(y_j)$ for all $j \in [n]$, it can exploit linear relations between different points on this line to recover sender elements beyond the intersection.

For example, let $\Delta \neq 0$ and fix i and distinct $j_1, j_2, j_3 \in [n]$. The receiver can compute

$$c := \frac{L_i(y_{j_1}) - L_i(y_{j_2})}{L_i(y_{j_2}) - L_i(y_{j_3})} = \frac{(K_i + y_{j_1}\Delta) - (K_i + y_{j_2}\Delta)}{(K_i + y_{j_2}\Delta) - (K_i + y_{j_3}\Delta)} = \frac{y_{j_1} - y_{j_2}}{y_{j_2} - y_{j_3}} \in \mathbb{F}.$$

Suppose now that $y_{j_1}, y_{j_2} \in X \cap Y$, so the receiver knows their values (they are equal to some $x_{i_1}, x_{i_2} \in X$). Then c is completely determined by known quantities, and the above equation becomes $c = \frac{y_{j_1} - y_{j_2}}{y_{j_2} - y_{j_3}}$, from which the receiver can solve for the *unknown* $y_{j_3} = y_{j_2} - \frac{y_{j_1} - y_{j_2}}{c}$.

Thus, given two matching sender vectors, the receiver can recover y_{j_3} for any unmatched vector. This reveals a sender element that is not in the intersection. This violates the PSI privacy requirement, which should guarantee that the receiver learns nothing about $Y \setminus (X \cap Y)$.

4.2. Flash: Efficient VOLE-Based PSI

The toy construction above illustrates that exposing the raw linear values $K_i + y_j\Delta$ is both inefficient and fundamentally insecure. The receiver can exploit their algebraic structure to reconstruct elements of $Y \setminus (X \cap Y)$. A key observation is that we can break this harmful linear structure by passing the VOLE-derived values through a random oracle. In particular, instead of sending $K_i + y_j\Delta$ directly, the sender only reveals pseudorandom tags obtained by hashing these values. Building on this idea, we further combine VOLE with Cuckoo hashing so that equality tests are performed only on a linear-size table, yielding our final protocol with $O(n)$ communication and computation.

Here we present our protocol Π_{Flash} with semi-honest security, which is shown in Figure 4.

Full Protocol: Π_{Flash}

- **Parties:** Two parties Receiver and Sender.
- **Parameters:** a security parameter λ, a finite field $\mathbb{F}$, a Cuckoo hashing parameter ε, and a fixed number of hash functions $b \in \{2, 3\}$. A random oracle $H : \{0,1\}^* \to \{0,1\}^\ell$, where ℓ is the tag length.
- **Input:** For public n, the receiver holds $X = \{x_1, \ldots, x_n\} \subseteq \mathbb{F}$, and the sender holds $Y = \{y_1, \ldots, y_n\} \subseteq \mathbb{F}$.

---

1. Receiver and sender agree on the Cuckoo hashing parameters. They run $(m, h_1, \ldots, h_b) \leftarrow \mathsf{CH.Setup}(\lambda, n, \varepsilon)$. The table size m and hash functions $h_1, \ldots, h_b$ are public.
2. Receiver inserts X into a Cuckoo table $T \leftarrow \mathsf{CH.Hash}(h_1, \ldots, h_b, X)$ using the hash functions. Record occupied positions and fill empty positions with independent uniform elements of $\mathbb{F}$.
3. Receiver sends $(\mathsf{Receiver}, \mathsf{sid})$ and sender sends $(\mathsf{Sender}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{rVOLE}}$. Then receiver gets $\vec{R} = (r_1, \ldots, r_m), \vec{M} = (M_1, \ldots, M_m)$, the sender gets $\vec{K} = (K_1, \ldots, K_m), \Delta$ such that $\forall i \in [m], M_i = K_i + r_i\Delta$. If the malicious rVOLE subprotocol aborts on this invocation, the honest party outputs $\perp$ and halts.
4. For each table position $i \in [m]$:
   (a) Receiver computes $e_i := r_i - T[i]$, and sends e_i to sender.
   (b) Sender updates its share for position i as $K_i := K_i + e_i\Delta$.
5. For each $y \in Y$ and each hash index $j \in [b]$, sender:
   (a) Compute $i_j := h_j(y) \in [m]$, and compute $Z_j(y) := K_{i_j} + y \cdot \Delta \in \mathbb{F}$.
   (b) Derive a tag $L_j(y) := H(i_j \,\|\, y \,\|\, Z_j(y)) \in \{0,1\}^\ell$.

   For each $j \in [b]$, let $\mathcal{M}_j := \{L_j(y) : y \in Y\}$ be the multiset of tags corresponding to hash function h_j. Sender sends $(\mathcal{M}_1, \ldots, \mathcal{M}_b)$ to receiver.
6. For each originally occupied position i, receiver computes its tag $U[i] := H(i \,\|\, T[i] \,\|\, M[i])$, checks whether $U[i] \in \bigcup_j \mathcal{M}_j$. If so, the receiver marks $T[i]$ as belonging to the intersection.
7. Let $I \subseteq X$ be the set of elements marked in the previous step. Receiver outputs I.

Figure 4. Protocol Π_{Flash} which realize the private set intersection functionality $\mathcal{F}_{\mathsf{PSI}}$. The gray box highlights the additional abort condition used in the maliciously secure variant.

Definition 1. *We say a protocol* Π *for PSI functionality* $\mathcal{F}_{\mathsf{PSI}}$ *is secure for PPT adversary* $\mathcal{A}$*, there exists a PPT simulator* Sim *such that,*

$$|Pr[REAL_{\Pi,\mathcal{A}}] - Pr[IDEAL_{\mathcal{F},\mathsf{Sim}}]| \leq \epsilon \tag{1}$$

where ϵ *is negligible.*

Theorem 1. Π_{Flash} *securely computes* $\mathcal{F}_{\mathsf{PSI}}$ *against semi-honest adversaries in the random oracle,* $\mathcal{F}_{\mathsf{rVOLE}}$*-hybrid model.*

Proof. Let $\mathsf{Bad}_{\mathsf{CH}}$ be failure of the first Cuckoo insertion trial, with probability at most p_{CH}. The receiver simulator knows X and runs the same retry procedure. The sender simulator uses a fresh setup seed and simulates first-trial

success. Coupling these executions changes the public setup transcript only on $\mathsf{Bad}_{\mathsf{CH}}$, contributing at most p_{CH} statistical distance. Truncating the receiver simulator's retry loop after λ trials adds at most $p_{\mathsf{CH}}^{\lambda}$ statistical distance.

Correctness. Assume both parties are honest. We show that, except with negligible probability, the receiver's output I equals $X \cap Y$.

First, by the guarantees of Cuckoo hashing, CH.Setup and CH.Hash succeed in inserting all elements of X into a table $T[1..m]$ using hashes $h_1, \ldots, h_b$ except with negligible probability. For each $x \in X$ let $i_x \in [m]$ be the position where x is finally stored and let $j_x \in [b]$ be such that $i_x = h_{j_x}(x)$, then $T[i_x] = x$.

Fix any $x \in X \cap Y$. In Step 5 the sender processes $y = x$ and, for j_x with $i_{j_x} = h_{j_x}(x) = i_x$, computes $Z_{j_x}(x) = K_{i_x} + x\Delta = M_{i_x}$ and then the tag $L_{j_x}(x) := H(i_x \parallel x \parallel Z_{j_x}(x)) = H(i_x \parallel x \parallel M_{i_x})$. In Step 6 the receiver computes $U[i_x] := H(i_x \parallel T[i_x] \parallel M_{i_x}) = H(i_x \parallel x \parallel M_{i_x})$, hence $U[i_x] = L_{j_x}(x)$ and so $U[i_x] \in \mathcal{M}_{j_x}$. Thus x is marked and belongs to the output set I. Therefore, every element of $X \cap Y$ is included in I.

Conversely, take any $x \in X \setminus Y$ and its position i_x and index j_x as above. The receiver computes $U[i_x] = H(i_x \parallel x \parallel M_{i_x})$. For any $y \in Y$ and $j \in [b]$, the sender's tag is $L_j(y) = H(h_j(y) \parallel y \parallel Z_j(y))$ with $Z_j(y) = K_{h_j(y)} + y\Delta$. If the receiver incorrectly marks x, then $U[i_x] = L_j(y)$ for some y, j. The inputs to H are distinct because $x \neq y$, so the two tags collide with probability $2^{-\ell}$. Thus, an $x \in X \setminus Y$ is marked only with negligible probability.

Accounting for Cuckoo insertion failure and the collision bound $mbn2^{-\ell}$ over all tag comparisons, we conclude that the receiver's output I equals $X \cap Y$ except with negligible probability for suitable ℓ.

Corrupted Receiver. We now consider the case where the receiver is corrupted, and the sender is honest. Let Sim_R be a PPT simulator that interacts with the ideal functionality $\mathcal{F}_{\mathsf{PSI}}$ and the corrupted receiver $\mathcal{A}_R$. We then describe the behavior of Sim_R.

- Sim_R maintains an initially empty table T. On any query z to H issued by $\mathcal{A}_R$, if $z \in \mathsf{T}$ then Sim_R returns the stored value, otherwise it samples $w \xleftarrow{\$} \{0,1\}^{\ell}$, stores (z, w) in T, and returns w.
- Sim_R plays the role of $\mathcal{F}_{\mathsf{rVOLE}}$. Whenever the corrupted receiver sends $(\mathsf{Receiver}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{rVOLE}}$, Sim_R simply chooses $(\vec{R}, \vec{M}) \xleftarrow{\$} (\mathbb{F}^2)^m$ and returns this pair to $\mathcal{A}_R$.
- Sim_R sends X to $\mathcal{F}_{\mathsf{PSI}}$ on behalf of the corrupted receiver and receives the intersection $I = X \cap Y$. For each occupied position i with $T[i] \in I$, the simulator computes the would-be local tag $U[i] := H(i \parallel T[i] \parallel M_i)$ by querying H. It then inserts $U[i]$ into $\mathcal{M}_{j_i}$ for the index j_i satisfying $h_{j_i}(T[i]) = i$.
  The simulator fills each multiset $\mathcal{M}_j$ to n entries with independent uniform strings in $\{0,1\}^{\ell}$. Finally, Sim_R sends $(\mathcal{M}_1, \ldots, \mathcal{M}_b)$ to $\mathcal{A}_R$.

  It remains to argue that the simulated view is indistinguishable from the real one. We use a sequence of hybrids.
  Hybrid H_0. This is the real execution of Π_{Flash}.
  Hybrid H_1. As H_0, except that the receiver's rVOLE outputs $(\vec{R}, \vec{M})$ are not generated by $\mathcal{F}_{\mathsf{rVOLE}}$ but are sampled uniformly at random as in $(\mathbb{F}^2)^m$. Since $(\vec{R}, \vec{M})$ is uniform and independent of Δ in the real protocol, taking $K_i = M_i - r_i\Delta$ before the sender update makes H_0 and H_1 identically distributed.
  Hybrid H_2. As H_1, except that the message $(\mathcal{M}_1, \ldots, \mathcal{M}_b)$ sent by the honest sender is replaced by the values generated by Sim_R in the third item above. We show that H_1 and H_2 are indistinguishable in the random-oracle model. Fix the receiver's randomness, the outputs $(\vec{R}, \vec{M})$, and the Cuckoo table T. In Hybrid H_1, for every $y \in Y$ and $j \in [b]$ the sender computes

$$Z_j(y) = K[h_j(y)] + y \cdot \Delta \in \mathbb{F}, \qquad L_j(y) = H(h_j(y) \parallel y \parallel Z_j(y)).$$

  For each $x \in X \cap Y$, there is exactly one pair (y, j) with $y = x$ and $h_j(y)$ equal to the position of x in the table, and for this pair we have $Z_j(y) = M_i$ and therefore $L_j(y) = H(i \parallel T[i] \parallel M_i) = U[i]$. This is exactly the relation that Sim_R enforces for $T[i] \in I$ in Hybrid H_2.

  Consider now any position i such that $T[i] \notin I$. With independent uniform dummy entries, the probability that any dummy belongs to Y is at most $(m - |X|)|Y|/|\mathbb{F}|$. Outside this event, $T[i] \notin Y$, so no sender input has the tuple $(i, T[i], M_i)$. Hence, the input tuple $(i, T[i], M_i)$ to the random oracle is distinct from all sender inputs $(h_j(y), y, Z_j(y))$. By the random-oracle assumption, the values

$$U[i] = H(i \parallel T[i] \parallel M_i) \quad \text{and} \quad \{L_j(y)\}_{y \in Y,\, j \in [b]}$$

  are evaluated at distinct oracle inputs. In Hybrid H_2, the simulator replaces the unmatched tags with independent uniform strings. For $i = h_j(y)$ we have $Z_j(y) = M_i + (y - T[i])\Delta$. In H_2, take Δ uniform and independent of the transcript. Given that transcript, each query (i, u, z) with $u \neq T[i]$ can hit an unmatched tag input only for

the unique Δ satisfying $z = M_i + (u - T[i])\Delta$. Thus the hit probability over all q_H queries is at most $q_H/|\mathbb{F}|$. Unless a hit or a dummy match occurs, the unmatched tags in the two hybrids can use the same independent oracle values, so H_1 and H_2 are indistinguishable up to these two bounds. Finally, by construction, H_2 is exactly the ideal-world execution with Sim_R. Therefore, the view of a semi-honest corrupted receiver in the real protocol is indistinguishable from its view in the ideal execution with $\mathcal{F}_{\mathsf{PSI}}$, as required.

Corrupted Sender. We now consider the case where the sender is corrupted and the receiver is honest. Let $\mathcal{A}_S$ be a semi-honest adversary controlling the sender, and let Sim_S be a PPT simulator that submits Y to $\mathcal{F}_{\mathsf{PSI}}$ and interacts with $\mathcal{A}_S$. We then describe the behavior of Sim_S.

- Sim_S maintains an initially empty table T. Whenever $\mathcal{A}_S$ queries H on some input z, the simulator checks if $z \in \mathsf{T}$. If so, it returns the stored value, otherwise it samples $w \xleftarrow{\$} \{0,1\}^\ell$, stores (z, w) in T, and returns w.
- Whenever corrupted sender sends $(\mathsf{Sender}, \mathsf{sid})$, Sim_S samples $\Delta \xleftarrow{\$} \mathbb{F}$ and independent $\vec{K} \xleftarrow{\$} \mathbb{F}^m$ for all $i \in [m]$, and hands these to $\mathcal{A}_S$.
- Sim_S replaces $e_i := r_i - T[i]$ for each i with uniformly random element in $\mathbb{F}$.

We compare the sender's view in the real and simulated executions using a short sequence of hybrids.

Hybrid H_0. The real execution of Π_{Flash} with honest receiver and corrupted sender $\mathcal{A}_S$.

Hybrid H_1. As H_0, except that the sender's rVOLE outputs (K_i, Δ) are not provided by $\mathcal{F}_{\mathsf{rVOLE}}$ but are sampled uniformly as in the second item above. Since $(\vec{K}, \Delta)$ is already uniform in $\mathbb{F}^{m+1}$ in H_0, the sender's views in H_0 and H_1 are identically distributed.

Hybrid H_2. As H_1, except that the values e_i received from the receiver are replaced by independent uniform elements $e_i \xleftarrow{\$} \mathbb{F}$ as in the third item. In H_1, the independent uniform masks r_i make the vector $(e_i)_i = (r_i - T[i])_i$ uniform given the sender's view before receiving it. Therefore, the sender's views in H_1 and H_2 are identically distributed.

By correctness, replacing the honest receiver's real output by $X \cap Y$ changes the joint distribution by at most the correctness error. The hybrids then simulate the sender's view up to the setup error, giving the ideal joint distribution. □

4.3. Flash with Malicious Security

We show that Π_{Flash} can be upgraded from semi-honest to malicious security by instantiating $\mathcal{F}_{\mathsf{rVOLE}}$ with any maliciously secure implementation that provides the same interface.

The key point is that, for a malicious receiver, security must be defined with respect to the effective input induced by its messages rather than any input it may claim to hold: a corrupted receiver is free to deviate from the honest algorithm and thus can arbitrarily change the x_i it commits to during the protocol. To simulate such a receiver in the $\mathcal{F}_{\mathsf{PSI}}$-ideal world, the simulator must therefore extract this real input; in our setting this is straightforward, because for each position i the simulator sees both the random-VOLE output r_i (from $\mathcal{F}_{\mathsf{rVOLE}}$) and the adversarial message e_i, and can recover the committed value as $x_i = r_i - e_i$. Against a malicious sender, Flash guarantees the privacy of the honest receiver's input.

Theorem 2. $\Pi_{\mathsf{Flash_Malicious}}$ *securely computes* $\mathcal{F}_{\mathsf{PSI}}$ *against malicious receivers and guarantees receiver-input privacy against malicious senders in the random oracle,* $\mathcal{F}_{\mathsf{rVOLE}}$*-hybrid model.*

Proof. Correctness. With honest parties, $\Pi_{\mathsf{Flash_Malicious}}$ has the same message flow as Π_{Flash} in the $\mathcal{F}_{\mathsf{rVOLE}}$-hybrid model, so correctness follows from Theorem 1. In the concrete protocol, an abort in the malicious rVOLE subprotocol terminates the PSI execution.

Corrupted sender. Let $\mathcal{A}_S$ corrupt the sender and let Sim_S be the simulator. The simulator preserves the sender's chosen rVOLE values $(\Delta, \vec{K})$, lazily simulates the random oracle, and replaces the receiver messages e_i with uniform field elements. Conditioned on the public setup and these chosen values, r_i remains uniform and hidden, so $e_i = r_i - T[i]$ is uniform independently of $T[i]$. The receiver sends no further messages. Thus the sender's view is independent of the receiver's input, up to the negligible setup error already accounted for, proving receiver-input privacy.

Corrupted receiver. Let $\mathcal{A}_R$ corrupt the receiver and let Sim_R be the simulator. Unlike in the semi-honest setting, a malicious receiver may arbitrarily deviate when computing the Cuckoo table or the values e_i, so the simulator cannot rely on the declared input X. Instead, in the $\mathcal{F}_{\mathsf{rVOLE}}$-hybrid model the simulator uses the messages to *extract* an effective input.

For each table position $i \in [m]$, $\mathcal{F}_{\mathsf{rVOLE}}$ delivers r_i and M_i to $\mathcal{A}_R$, and the receiver sends some field element e_i to the sender. Regardless of how $\mathcal{A}_R$ chooses e_i, the pair (r_i, e_i) uniquely defines a value $T[i]^\star := r_i - e_i \in \mathbb{F}$,

which we interpret as the content of position i in the (possibly malformed) table chosen by the receiver. Let $X^\star$ be the set of distinct values $T[i]^\star$. Hence $|X^\star| \le m$. For Flash, we set $n_X = n_Y = n$ and $n'_X = m$. Sim_R submits $X^\star$ as the receiver's input to $\mathcal{F}_{\mathsf{PSI}}$ and obtains the intersection $I = X^\star \cap Y$.

Given the receiver's rVOLE values and messages, the sender's updated share at position i is $M_i - T[i]^\star\Delta$. The identifiable matching tags correspond to $X^\star$; the simulator retains the extracted table positions to reproduce their dependencies.

Given I and the collection $\{M_i\}_i$, the simulator inserts $H(i \parallel T[i]^\star \parallel M_i)$ into $\mathcal{M}_j$ whenever $T[i]^\star \in I$ and $h_j(T[i]^\star) = i$, and fills each multiset to n entries as in Theorem 1. The same oracle-query bound applies with $T[i]^\star$ in place of $T[i]$; Δ is uniform and independent of the receiver in the random-tag hybrid. Thus, the simulated view is indistinguishable from the real one. □

5. Evaluations

5.1. Asymptotic Analysis

We next analyze the asymptotic efficiency of Π_{Flash}. For an input set of size n and Cuckoo parameters (ε, b), the Cuckoo setup outputs a table size $m = (1+\varepsilon)n$ and hash functions $h_1, \ldots, h_b : \mathbb{F} \to [m]$. Standard analysis guarantees that insertion fails only with probability $\mathsf{negl}(\lambda)$.

We first analyze the computation overhead. The protocol invokes a single call to $\mathcal{F}_{\mathsf{rVOLE}}$ that outputs $m = (1+\varepsilon)n$ random VOLE correlations in one batch. We instantiate $\mathcal{F}_{\mathsf{rVOLE}}$ with PCG-based random VOLE [30]. For fixed security and block-accumulate code parameters, generating m correlations requires $O(m)$ operations, including base-OT setup and any malicious consistency checks. On top of this, the work done by our PSI layer is also linear in n. On the receiver side, Cuckoo insertion of n elements into a table of size $m = (1+\varepsilon)n$; this requires a constant number of evaluations of $(h_1, \ldots, h_b)$ per element and at most a constant number of relocations per insertion. For each table position $i \in [m]$, one field subtraction $e_i = r_i - T[i]$ and one random-oracle evaluation $U[i] = H(i \parallel T[i] \parallel M_i)$. On the sender side, for each $y \in Y$ and each $j \in [b]$, one hash evaluation $i_j = h_j(y)$, one field multiplication-and-addition $Z_j(y) = K_{i_j} + y \cdot \Delta$, and one random-oracle evaluation $L_j(y) = H(h_j(y) \parallel y \parallel Z_j(y))$. Summing up, apart from the single rVOLE call generating $(1+\varepsilon)n$ correlations, the PSI layer performs at most $(1+\varepsilon)n$ field subtractions on the receiver, $2bn$ field multiplication/additions on the sender, and $(m+bn) = (1+\varepsilon+b)n$ calls to hash functions. The expected total computation is $O((1+\varepsilon)n)+O((1+\varepsilon+b)n)$ for fixed ε, b, and security parameters, where the first term is the full rVOLE-generation cost.

We then analyze the communication overhead. For the communication complexity, we separate the cost of the rVOLE primitive from the cost of our PSI layer. For this instantiation and fixed security parameters, the full rVOLE communication is $O(\log m)$ bits: a constant number of sparse noise positions requires $O(\log m)$ OTs. This bound includes base-OT setup and any malicious consistency checks. Then, Π_{Flash} sends only two types of messages. From receiver to sender, the $m = (1+\varepsilon)n$ field elements e_i, each encoded on κ bits. This contributes $(1+\varepsilon)\kappa n$ bits. From sender to receiver, the b tag multisets $(\mathcal{M}_1, \ldots, \mathcal{M}_b)$, which together contain bn tags of length ℓ bits. This contributes $b\ell n$ bits. Hence the total communication is $O(\log m) + m\kappa + bn\ell + O(\kappa)$ bits, where the first term is the full rVOLE communication and the final term accounts for the public hash seed.

5.2. Implementation Analysis

We implement Π_{Flash} in C++. All experiments are run on a machine equipped with an Intel Core i9-10900 CPU, 64 GB of RAM, and Ubuntu 22.04 LTS. Unless otherwise stated, all protocols are implemented in a single thread and executed between two logical parties communicating over TCP sockets. We evaluate three representative network configurations. LAN emulates a local network with 10 Gbps bandwidth and 0.2 ms round-trip time (RTT). 1 Gbps emulates a wide-area link with a nominal rate of 1 Gbps and 20 ms RTT, while 100 Mbps uses a nominal rate of 100 Mbps and the same 20 ms RTT. We use Linux `tc` to set these rates, with HTB burst allowances of up to 15 MiB. We report end-to-end protocol runtimes.

We set the statistical security parameter to $\lambda = 40$ and the computational security parameter to $\kappa = 128$. The `cryptoTools` `CuckooIndex` implementation derives three distinct candidate positions from $\mathrm{AES}_s(x) \oplus x$, where s is a public 128-bit seed. For the cuckoo-related parameters, we follow the recommendation of [31]: for a set size n we choose the cuckoo table size m according to the empirical parameter fit $\lambda = a_n \frac{m}{n} + b_n$, where $a_n \approx 123.5$ and $b_n \approx -130 - \log n$. Our implementation discards the partial table on insertion failure and repeats Cuckoo setup and insertion with fresh hash functions until successful, using the same table size. The required single-trial failure probability is negligible, with concrete target 2^{-40}.

We compare our protocol against prior fast PSI schemes, including DCW13 [15], KKRT16 [18], RR17a [32], RR17b [33], CM20 [21], RR22 [9], and PRTY20 [6], where RR22 [9] is the state-of-the-art scheme. For each scheme, we measure both the end-to-end running time and the total communication in megabytes under the stated network configurations. For DCW13 [15], KKRT16 [18], RR17a [32], and RR17b [33] we use the implementations from the `libPSI` framework (https://github.com/osu-crypto/libPSI (accessed on 25 June 2026)), for CM20 [21] we rely on the implementation from `OPRF-PSI` (https://github.com/peihanmiao/OPRF-PSI (accessed on 25 June 2026)), and for RR22 [9] we use the `volepsi` implementation (https://github.com/ladnir/volepsi (accessed on 25 June 2026)). For PRTY20 [6], we use the `PaXoS_PSI` implementation (https://github.com/cryptobiu/PaXoS_PSI (accessed on 25 June 2026)).

In each column of Tables 1 and 2, we rank the semi-honest and malicious results separately. Tied values share the same shading, and the same shade can appear in both categories. Light gray marks the second-lowest distinct value in each category.

Table 1. Communication and LAN running time of PSI protocols. "–" indicates that the experiment ran out of memory. Within each security category, medium-gray and light-gray shading mark the best and second-best results, respectively. RR17a-sm denotes the standard-model variant.

Protocol	Security	Comm (MB)				LAN (ms)			
		2^{12}	2^{16}	2^{20}	2^{24}	2^{12}	2^{16}	2^{20}	2^{24}
DCW13 [15]	Semi-honest	12.16	194.5	3112	–	69	1043	22,608	–
KKRT16 [18]	Semi-honest	0.43	6.9	114.3	1893	9	65	1533	26,708
RR17a [32]	Malicious	9.08	154.2	2603	–	66	1129	20,666	–
RR17a-sm [32]	Malicious	358.5	6074	–	–	760	12,083	–	–
RR17b [33]	Malicious	5.68	95.78	1616.03	–	42	502	9211	–
CM20 [21]	Semi-honest	0.355	5.343	85.16	1362	99	367	5162	109,757
PRTY20 [6]	Malicious	0.871	13.05	220.15	3464.13	364	502	4861	83,104
RR22 [9]	Semi-honest	0.726	2.664	33.95	550.6	33	57	698	15,200
	Malicious	0.761	3.105	40.24	634.5	35	62	814	17,481
Ours	Semi-honest	0.515	3.77	58.65	986.31	9	41	760	18,511
	Malicious	0.599	4.90	73.66	1178.31	10	46	809	19,542

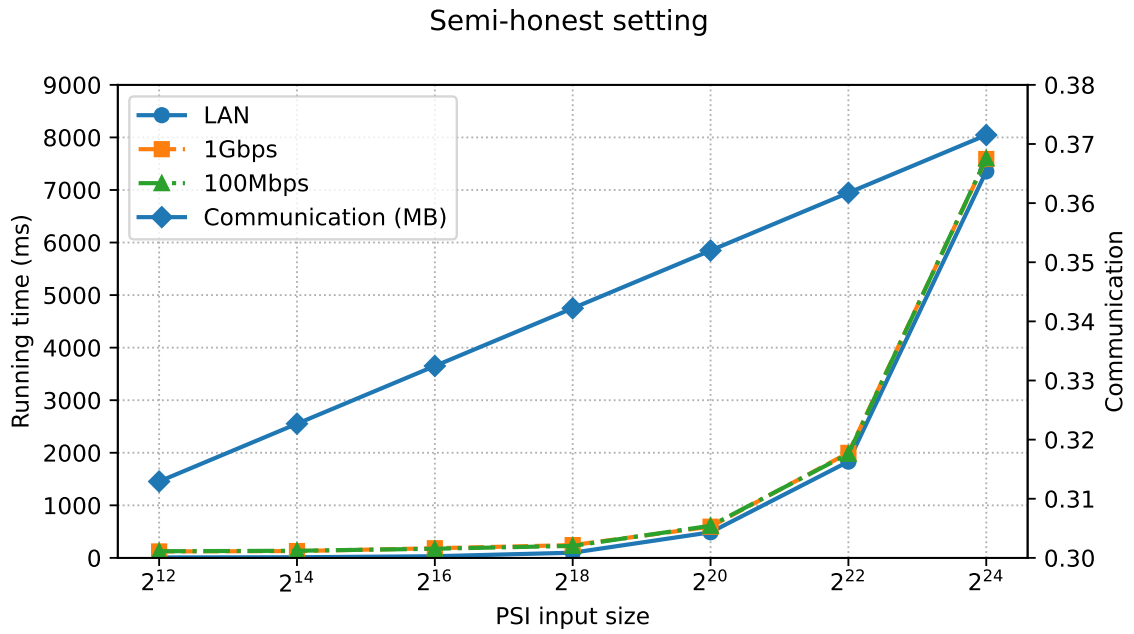
From Table 1 we first compare the communication of our protocol with prior works. For set sizes 2^{16}–2^{24}, our semi-honest variant has the second-lowest communication among the semi-honest rows: it is consistently close to CM20 [21] and RR22 [9], and strictly better than the older DCW13 [15], RR17a [32] and RR17b [33]. In particular, for small and medium sets (2^{12}–2^{20}), our semi-honest variant incurs only a modest overhead (at most about a factor of 1.73) compared to the semi-honest RR22 [9], while for 2^{24} elements our communication remains between RR22 [9] and CM20 [21]. For example, at 2^{24} our protocol sends about 986.31 MB compared to CM20 [21]'s 1362 MB (about 28% less), while staying within a factor 1.8 of RR22 [9]. The PSI overhead percentages are calculated from the values displayed in Tables 1 and 2. The communication increases by 16.31%, 29.97%, 25.59%, and 19.47% at 2^{12}, 2^{16}, 2^{20}, and 2^{24}, respectively.

Tables 1 and 2 report the running times of all protocols under LAN and WAN settings. On the LAN setting our semi-honest protocol is among the fastest for small and medium input sizes, and remains competitive even for 2^{24} elements. For input sizes 2^{12} and 2^{16}, it improves over the state-of-the-art RR22 [9] by about $3.7\times$ and $1.4\times$, respectively, and is nearly an order of magnitude faster than CM20 [21] (about $9\times$ at 2^{16}); for 2^{24} it is still around $6\times$ faster than CM20 [21]. The malicious version adds 5.57%–12.20% LAN runtime overhead across the four sizes and, for small sets, even outperforms the semi-honest RR22 [9]. Over a 1Gbps WAN setting, our semi-honest variant again attains the best performance for 2^{12} and 2^{16}, being roughly $2.1\times$ faster than RR22 [9] in both cases and about $3.5\times$ faster than CM20 [21] at 2^{16}, and it essentially matches RR22 [9] for 2^{20}. Only for the largest size 2^{24} does RR22 [9] become faster due to its lower bandwidth usage. On the 100Mbps setting, where communication dominates, our protocol still clearly outperforms earlier PSI schemes for small sets (e.g., about $2.3\times$ faster than RR22 [9] and about $4\times$ faster than CM20 [21] at 2^{16}), but becomes slower than RR22 [9] for 2^{20} and 2^{24}, reflecting the fact that we trade slightly higher bandwidth for lower computation.

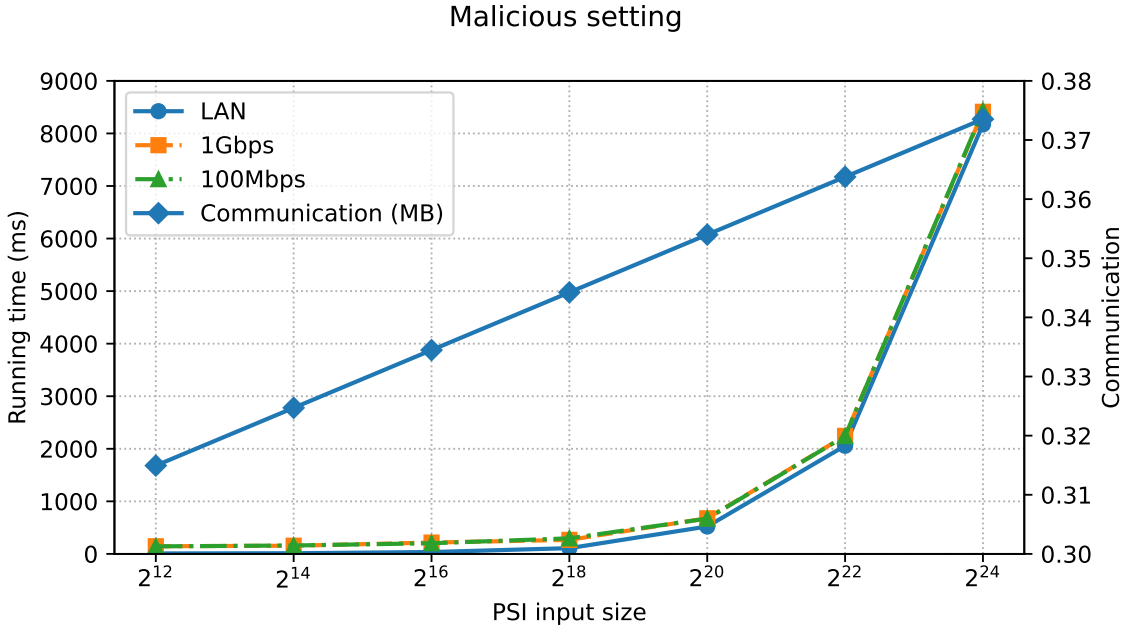
Figure 5 plots the running time and communication of the rVOLE in the semi-honest and malicious settings. On LAN the malicious variant is about 3%–14% slower than the semi-honest one (typically within 7%–14% for

medium and large input sizes), and over 1Gbps and 100Mbps settings the overhead is of the same order, bounded by roughly 30%. In contrast, the communication curves for the two variants are almost indistinguishable: the malicious protocol sends only an additional 0.002–0.003 MB per rVOLE instance, i.e., well below 1% extra bandwidth.

For the practically relevant sizes 2^{20} and 2^{24}, rVOLE contributes roughly 40%–64% of the running time on LAN and 27%–40% on the 1Gbps setting, and less than 13% on a 100Mbps link, while its communication is below 1% of the total PSI bandwidth. For 2^{20} and 2^{24} elements, the overall running time increase from semi-honest to malicious security is 5.57–15.96% on LAN and 1Gbps and up to 26.00% on 100Mbps, with communication increases ranging from 19.47% to 25.59%. For input sizes 2^{12}, 2^{16}, 2^{20}, and 2^{24}, the runtime increases (LAN, 1Gbps, 100Mbps) are $(11.11, 15.75, 12.08)\%$, $(12.20, 27.93, 26.39)\%$, $(6.45, 15.96, 26.00)\%$, and $(5.57, 7.93, 17.39)\%$, respectively.



(a)



(b)

Figure 5. Performance of rVOLE in different security settings. (**a**) Semi-honest setting; (**b**) Malicious setting.

Table 2. Running time of PSI protocols over 1Gbps and 100Mbps WAN links. "–" indicates that the experiment ran out of memory. Within each security category, medium-gray and light-gray shading mark the best and second-best results, respectively. RR17a-sm denotes the standard-model variant.

Protocol	Security	1 Gbps (ms)				100 Mbps (ms)			
		2^{12}	2^{16}	2^{20}	2^{24}	2^{12}	2^{16}	2^{20}	2^{24}
DCW13 [15]	Semi-honest	512	3503	56,489	–	1373	17,517	280,450	–
KKRT16 [18]	Semi-honest	207	1348	2124	30,100	224	1858	10,476	167,642
RR17a [32]	Malicious	492	2468	34,418	–	1174	13,555	227,846	–
RR17a-sm [32]	Malicious	5137	79,096	–	–	31,346	516,809	–	–
RR17b [33]	Malicious	477	1747	22,105	–	845	8789	139,668	–
CM20 [21]	Semi-honest	318	775.9	5405	113,015	341	862	8010	128,626
PRTY20 [6]	Malicious	555	894	7103	115,571	587	1801	23,406	371,547
RR22 [9]	Semi-honest	304	459	1496	17,583	324	496	2619	56,475
	Malicious	340	554	1623	23,046	369	569	3125	65,583
Ours	Semi-honest	146	222	1479	27,839	149	216	4839	98,405
	Malicious	169	284	1715	30,048	167	273	6097	115,516

6. Conclusions

In this work, we proposed a new VOLE-based paradigm for private set intersection and instantiated it with a concrete PSI construction. Our design cleanly separates correlated randomness generation from the protocol logic, which yields an efficient semi-honest protocol and allows upgrading to malicious security with average runtime and communication increases of 15.40% and 22.84%, respectively. Experiments under various network conditions show that our implementation is competitive with, and often faster than, the state of the art while keeping communication low. A current limitation is that the protocol is tailored to the two-party setting and does not directly handle many mutually distrustful clients. Extending our VOLE-based approach to scalable multi-client PSI is an direction for future work.

Author Contributions

Z.J.: conceptualization, formal analysis, investigation, methodology, software, and writing—original draft; X.F.: writing—review & editing. All authors have read and agreed to the published version of the manuscript.

Funding

This research received no external funding.

Institutional Review Board Statement

Not applicable.

Informed Consent Statement

Not applicable.

Data Availability Statement

Not applicable.

Conflicts of Interest

The authors declare no conflict of interest.

Use of AI and AI-Assisted Technologies

During the preparation of this work, the authors used ChatGPT to improve the grammar of the manuscript. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

References

1. Freedman, M.J.; Nissim, K.; Pinkas, B. Efficient Private Matching and Set Intersection. In *Advances in Cryptology—EUROCRYPT 2004, Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, 2–6 May 2004*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 1–19. https://doi.org/10.1007/978-3-540-24676-3_1.
2. Huang, Y.; Evans, D.; Katz, J. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? In Proceedings of the 19th Annual Network and Distributed System Security Symposium (NDSS 2012), San Diego, CA, USA, 5–8 February 2012.
3. Hetz, L.; Schneider, T.; Weinert, C. Scaling Mobile Private Contact Discovery to Billions of Users. In *Computer Security—ESORICS 2023, Proceedings of the 28th European Symposium on Research in Computer Security, the Hague, the Netherlands, 25–29 September 2023*; Springer: Cham, Switzerland, 2024; pp. 455–476. https://doi.org/10.1007/978-3-031-50594-2_23.
4. Lauter, K.; Kannepalli, S.; Laine, K.; et al. Password Monitor: Safeguarding Passwords in Microsoft Edge. Available online: https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/ (accessed on 13 January 2026).
5. Scheffler, S.; Mayer, J. SoK: Content Moderation for End-to-End Encryption. *Proc. Priv. Enhancing Technol.* **2023**, *2023*, 403–429. https://doi.org/10.56553/popets-2023-0060.
6. Pinkas, B.; Rosulek, M.; Trieu, N.; et al. PSI from PaXoS: Fast, Malicious Private Set Intersection. In *Advances in Cryptology—EUROCRYPT 2020, Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, 10–14 May 2020*; Springer: Cham, Switzerland, 2020; pp. 739–767. https://doi.org/10.1007/978-3-030-45724-2_25.
7. Garimella, G.; Pinkas, B.; Rosulek, M.; et al. Oblivious Key-Value Stores and Amplification for Private Set Intersection. In *Advances in Cryptology—CRYPTO 2021, Proceedings of the 41st Annual International Cryptology Conference, CRYPTO 2021, Online, 16–20 August 2021*; Springer: Cham, Switzerland, 2021; pp. 395–425. https://doi.org/10.1007/978-3-030-84245-1_14.
8. Rindal, P.; Schoppmann, P. VOLE-PSI: Fast OPRF and Circuit-PSI from Vector-OLE. In *Advances in Cryptology—EUROCRYPT 2021, Proceedings of the 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, 17–21 October 2021*; Springer: Cham, Switzerland, 2021; pp. 901–930. https://doi.org/10.1007/978-3-030-77886-6_31.
9. Raghuraman, S.; Rindal, P. Blazing Fast PSI from Improved OKVS and Subfield VOLE. In Proceedings of the CCS'22: 2022 ACM SIGSAC Conference on Computer and Communications Security, Los Angeles, CA, USA, 7–11 November 2022; pp. 2505–2517. https://doi.org/10.1145/3548606.3560658.
10. Bienstock, A.; Patel, S.; Seo, J.Y.; et al. Near-Optimal Oblivious Key-Value Stores for Efficient PSI, PSU and Volume-Hiding Multi-Maps. In Proceedings of the USENIX Security Symposium (USENIX Security), Anaheim, CA, USA, 9–11 August 2023; pp. 301–318.
11. Meadows, C. A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party. In Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 7–9 April 1986; pp. 134–137. https://doi.org/10.1109/sp.1986.10022.
12. Chen, H.; Laine, K.; Rindal, P. Fast Private Set Intersection from Homomorphic Encryption. In Proceedings of the CCS'17: 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1243–1255. https://doi.org/10.1145/3133956.3134061.
13. Ghosh, S.; Simkin, M. The Communication Complexity of Threshold Private Set Intersection. In Proceedings of the 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 2019; pp. 3–29. https://doi.org/10.1007/978-3-030-26951-7_1.
14. Hazay, C.; Lindell, Y. Efficient Protocols for Set Intersection and Pattern Matching with Security against Malicious and Covert Adversaries. In *Theory of Cryptography, Proceedings of the Fifth Theory of Cryptography Conference, TCC 2008, New York, NY, USA, 19–21 March 2008*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 155–175. https://doi.org/10.1007/978-3-540-78524-8_10.
15. Dong, C.; Chen, L.; Wen, Z. When Private Set Intersection Meets Big Data: An Efficient and Scalable Protocol. In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany, 4–8 November 2013; pp. 789–800. https://doi.org/10.1145/2508859.2516701.
16. Pinkas, B.; Schneider, T.; Zohner, M. Faster Private Set Intersection Based on OT Extension. In Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, 20–22 August 2014; pp. 797–812.
17. Pinkas, B.; Schneider, T.; Segev, G.; et al. Phasing: Private Set Intersection Using Permutation-Based Hashing. In Proceedings of the USENIX Security Symposium (USENIX Security), Washington, DC, USA, 12–14 August 2015; pp. 515–530.
18. Kolesnikov, V.; Kumaresan, R.; Rosulek, M.; et al. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. In Proceedings of the CCS'16: 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 818–829. https://doi.org/10.1145/2976749.2978381.
19. Pinkas, B.; Schneider, T.; Zohner, M. Scalable Private Set Intersection Based on OT Extension. *ACM Trans. Priv. Secur.*

2018, *21*, 7. https://doi.org/10.1145/3154794.

20. Pinkas, B.; Rosulek, M.; Trieu, N.; et al. SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension. In Proceedings of the Annual International Cryptology Conference (CRYPTO), Santa Barbara, CA, USA, 18–22 August 2019; pp. 401–431. https://doi.org/10.1007/978-3-030-26954-8_13.
21. Chase, M.; Miao, P. Private Set Intersection in the Internet Setting from Lightweight Oblivious PRF. In *Advances in Cryptology—CRYPTO 2020, Proceedings of the 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, 17–21 August 2020*; Springer: Cham, Switzerland, 2020; pp. 34–63. https://doi.org/10.1007/978-3-030-56877-1_2.
22. Mahdavi, R.A.; Lukas, N.; Ebrahimianghazani, F.; et al. PEPSI: Practically Efficient Private Set Intersection in the Unbalanced Setting. In Proceedings of the 33rd USENIX Security Symposium, Philadelphia, PA, USA, 14–16 August 2024; pp. 6453–6470.
23. Chielle, E.; Maniatakos, M. Recurrent Private Set Intersection for Unbalanced Databases with Cuckoo Hashing and Leveled FHE. In Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 24–28 February 2025. https://doi.org/10.14722/ndss.2025.240365.
24. Uzun, E.; Chung, S.P.; Kolesnikov, V.; et al. Fuzzy Labeled Private Set Intersection with Applications to Private Real-Time Biometric Search. In Proceedings of the 30th USENIX Security Symposium, Online, 11–13 August 2021; pp. 911–928.
25. Liu, Q.; Guo, X.; Yang, K.; et al. Labeled Private Set Intersection from Distributed Point Function. *IEEE Trans. Inf. Forensics Secur.* **2025**, *20*, 2970–2983. https://doi.org/10.1109/tifs.2025.3550059.
26. Gao, Y.; Luo, Y.; Wang, L.; et al. Efficient Scalable Multi-Party Private Set Intersection(-Variants) from Bicentric Zero-Sharing. In Proceedings of the CCS'24: ACM SIGSAC Conference on Computer and Communications Security, Salt Lake City, UT, USA, 14–18 October 2024; pp. 4137–4151. https://doi.org/10.1145/3658644.3690245.
27. Dong, M.; Chen, Y.; Zhang, C.; et al. Multi-Party Private Set Operations from Predicative Zero-Sharing. In Proceedings of the CCS'25: ACM SIGSAC Conference on Computer and Communications Security, Taipei, Taiwan, 13–17 October 2025; pp. 4079–4093. https://doi.org/10.1145/3719027.3765234.
28. Wu, M.; Yuen, T.H. Efficient Unbalanced Private Set Intersection Cardinality and User-Friendly Privacy-Preserving Contact Tracing. In Proceedings of the 32nd USENIX Security Symposium, Anaheim, CA, USA, 9–11 August 2023; pp. 283–300. https://dl.acm.org/doi/10.5555/3620237.3620254.
29. Yang, Y.; Yang, Y.; Chen, X.; et al. DMPSI: Efficient Scalable Delegated Multiparty PSI and PSI-CA with Oblivious PRF. *IEEE Trans. Serv. Comput.* **2024**, *17*, 497–508. https://doi.org/10.1109/tsc.2024.3356667.
30. Raghuraman, S.; Rindal, P.; Tanguy, T. Expand-Convolute Codes for Pseudorandom Correlation Generators from LPN. In *Advances in Cryptology—CRYPTO 2023, Proceedings of the 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, 20–24 August 2023*; Springer: Cham, Switzerland, 2023; pp. 602–632. https://doi.org/10.1007/978-3-031-38551-3_19.
31. Demmler, D.; Rindal, P.; Rosulek, M.; et al. PIR-PSI: Scaling Private Contact Discovery. *Proc. Priv. Enhancing Technol.* **2018**, *2018*, 159–178. https://doi.org/10.1515/popets-2018-0037.
32. Rindal, P.; Rosulek, M. Improved Private Set Intersection against Malicious Adversaries. In *Advances in Cryptology—EUROCRYPT 2017, Proceedings of the 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, 30 April– 4 May 2017*; Springer: Cham, Switzerland, 2017; pp. 235–259. https://doi.org/10.1007/978-3-319-56620-7_9.
33. Rindal, P.; Rosulek, M. Malicious-Secure Private Set Intersection via Dual Execution. In Proceedings of the CCS'17: 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1229–1242. https://doi.org/10.1145/3133956.3134044.