Scalable Private Set Intersection over Distributed Encrypted Data

Seunghun Paik
Hanyang University
Seoul, Republic of Korea
whitesoonguh@hanyang.ac.kr

Nirajan Koirala
University of Notre Dame
Notre Dame, IN, USA
nkoirala@nd.edu

Jack Nero
University of Notre Dame
Notre Dame, IN, USA
jnero@nd.edu

Hyunjung Son
Hanyang University
Seoul, Republic of Korea
dk9050rx@hanyang.ac.kr

Yunki Kim
Hanyang University
Seoul, Republic of Korea
yunki@hanyang.ac.kr

Jae Hong Seo*
Hanyang University
Seoul, Republic of Korea
jaehongseo@hanyang.ac.kr

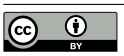
Taeho Jung*
University of Notre Dame
Notre Dame, IN, USA
tjung@nd.edu

Abstract

Finding intersections across sensitive data is a core operation in many real-world data-driven applications, such as healthcare, anti-money laundering, financial fraud, or watchlist applications. These applications often require large-scale collaboration across thousands or more independent sources, such as hospitals, financial institutions, or identity bureaus, where all records must remain encrypted during storage and computation, and are typically outsourced to dedicated/cloud servers. Such a highly distributed, large-scale, and encrypted setting makes it very challenging to apply existing solutions, e.g., (multi-party) private set intersection (PSI) or private membership test (PMT).

In this paper, we present Distributed and Outsourced PSI (DO-PSI), an efficient and scalable PSI protocol over outsourced, encrypted, and highly distributed datasets. Our key technique lies in a generic threshold fully homomorphic encryption (FHE) based framework that aggregates equality results additively, which ensures high scalability to a large number of data sources. In addition, we propose a novel technique called *nonzero-preserving mapping*, which maps a zero vector to zero and preserves nonzero values. This allows homomorphic equality tests over a smaller base field, substantially reducing computation while enabling higher-precision representations. We implement DO-PSI and conduct extensive experiments, showing that ours substantially outperforms existing methods in both computation and communication overheads. Our protocol handles a billion-scale set distributed and outsourced to a thousand data owners within one minute, directly reflecting large-scale deployment scenarios, and achieves up to an 11.16× improvement in end-to-end latency over prior state-of-the-art methods.

*Co-corresponding authors



ASIA CCS '26, Bangalore, India

ACM ISBN 979-8-4007-2356-8/2026/06
https://doi.org/10.1145/3779208.3785272

CCS Concepts

• **Security and privacy → Privacy-preserving protocols**; **Management and querying of encrypted data**.

Keywords

Private Set Intersection, Encrypted Datasets, FHE

ACM Reference Format:
Seunghun Paik, Nirajan Koirala, Jack Nero, Hyunjung Son, Yunki Kim, Jae Hong Seo, and Taeho Jung. 2026. Scalable Private Set Intersection over Distributed Encrypted Data. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '26), June 01–05, 2026, Bangalore, India.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3779208.3785272

1 Introduction

In today's digital landscape, sensitive information is often gathered, distributed, and processed among many entities. Many industries, including finance, healthcare, and government organizations, often require balancing privacy with usability. Private Set Intersection (PSI), which enables two parties to learn only the intersection of their respective input sets without revealing non-matching items, has been widely deployed to achieve this balance. It has become a core building block in a wide range of privacy-sensitive data ecosystems, including private contact discovery [26, 83], password-breach checks [39], location sharing, malware signature checking, and advertising conversion measurement [15, 16, 21]. Beyond these one-shot examples, PSI often serves as a pre-processing step for privacy-preserving record linkage and downstream analytics, where organizations first use PSI to link records referring to the same entity across silos and then perform additional computations only on the linked subset, thereby minimizing disclosure and computation on non-matches [15, 21].

Many real-world scenarios for PSI-applications are asymmetric, i.e., a server (or service provider) holds a large, relatively static set (e.g., credential dumps, malware signatures, watchlists), and many clients hold comparatively small sets (e.g., a user's contacts, installed apps) that they wish to check against the server [21]. Consequently, several PSI protocols are optimized for this asymmetric setting, aiming to (i) allow the server to perform heavy work once in an offline

phase and (ii) keep each per-client online run nearly linear in the client's set size while preserving strong privacy [15, 16, 21]. These designs underlie applications such as password-breach checking, private contact discovery at scale, and mobile malware scanning, where the server's database dwarfs individual client inputs [15, 16]. In particular, a special case of asymmetric PSI where the client's set has a single element, often referred to as Private Membership Test (PMT), has also become a dominant primitive in practical applications such as password leak checks and watchlist screening [32, 81].

Furthermore, collaborative analysis of sensitive data in real-world scenarios often spans multiple organizations and terabyte-scale datasets. For example, in healthcare, data is siloed and highly fragmented across hospitals, clinics, and networks owned by different entities. The United States alone has more than 6,000 hospitals [7] maintaining their own patient databases. The finance domain is another example where vast networks of data owned by banks, credit unions, insurance companies, and market institutions are engaged. There are over 11,000 financial institutions globally connected via the SWIFT network [70] (each with its own transaction and account databases), and thousands in individual countries as well (the United States alone has around 4,000 banks and 5,000 credit unions [75]). In addition, these entities increasingly offload storage and analytics to dedicated servers/cloud [24, 27, 37, 68], where regulations such as GDPR and HIPAA obligate them to keep all outsourced customer data encrypted [57, 80]. For stronger regulatory compliance and individual privacy, sensitive data must be encrypted at the point of collection and remain encrypted throughout storage, distribution, and computation [12, 22, 53, 58, 78].

Such a large, distributed, *encrypted* ecosystem poses multiple challenges to employing current PSI protocols. First, many state-of-the-art designs assume that datasets are available in plaintext or amenable to plaintext preprocessing for their protocol design or core optimizations [21, 52, 83, 85], without which performance degrades significantly. Second, scaling to *many* independent data sources is costly as aggregation of queried results from a large number of distributed data sources often requires significant computation and communication overheads [15, 16, 21, 83]. For instance, polynomial-evaluation approaches to test $y \in X$, evaluates $f(x) = \prod_{z \in X}(x-z)$ to confirm if $f(y) = 0$ scale poorly across many owners because multi-party aggregation requires the outputs to be multiplied, yielding prohibitive computation and/or communication at crowd scale [2, 15, 16, 21, 30, 74, 83, 85]. Third, even though some solutions like delegated PSI centralize encrypted datasets at a cloud service [1–3, 26, 87]; these systems presume a powerful aggregator that can ingest and manage all participants' ciphertexts and typically do not scale cleanly in either dataset size or the number of owners. Fourth, recent multi-holder PMT systems [19, 32, 81] either disclose membership outcomes to a central computing entity and/or incur costs that rise quickly with the number of participants. Finally, recently proposed PSMT (Private Segmented Membership Test) [45] partially addresses this setting by segmenting data across owners and achieves high owner scalability, but it supports limited set sizes, exhibits a non-negligible false-positive rate, incurs high computation/communication, and requires non-overlapping sets, which is often unrealistic in practice.

In conclusion, despite the emerging needs for the PSI protocols tailored for privacy-sensitive data ecosystems, to our knowledge, no PSI protocol fully addresses the PSI over highly distributed datasets that are encrypted and outsourced to a remote party for delegation. In particular, such a restrictive setting prevents the direct use of prominent cryptographic building blocks commonly used for designing PSI protocols, including oblivious transfer (OT) [65], oblivious PRF (OPRF) [29], oblivious key-value store (OKVS) [33], and private information retrieval (PIR) [20], because they all require plaintext access to the data. These limitations call for a new PSI protocol that (i) operates directly over encrypted datasets, (ii) scales across thousands of independent and distributed owners, and (iii) remains efficient even with large sets and elements sizes.

Our Contribution. We present Distributed and Outsourced Private Set Intersection (DO-PSI), a novel PSI protocol to simultaneously support encrypted, outsourced, and highly distributed sets, thus fully addressing the limitations of prior approaches. Our protocol utilizes threshold fully homomorphic encryption (FHE), which facilitates the design of communication- and round-efficient secure multi-party computation protocols [6, 8, 45], while enabling data owners to outsource their data in an encrypted form to remote servers (e.g., cloud services). We introduce two core techniques to build an efficient and highly scalable PSI protocol supporting encrypted, outsourced sets distributed to thousands of servers. First, we propose a generic, highly scalable framework for DO-PSI based on threshold FHE, which supports efficient aggregation of membership results from servers holding outsourced data. Our core insight is to leverage a homomorphic equality circuit, i.e., returning 1 for identical inputs and 0 for otherwise. This enables cheap addition-only result aggregation, hence avoiding the FHE parameter blow-up from the growing number of servers, unlike previous polynomial-evaluation approaches [2, 15, 16, 21, 83].

The main challenge of this approach is to design an efficient homomorphic equality evaluation method. In particular, to ensure negligible false positives from the collision between items, it should support a sufficiently large domain, e.g., 80-bit or 128-bit integers. To address this, we introduce a novel technique, called the nonzero-preserving mapping (NPM), which enables us to compress a vector $\vec{v}$ to a scalar value w while ensuring that $w = 0$ if and only if $\vec{v} = \vec{0}$. This property enables us to handle long set items by viewing them as vectors defined over a smaller field, e.g., $\mathbb{F}_p$ with $p = 2^{16} + 1$; designing an equality circuit for a much smaller domain suffices. With these techniques and several optimisations, we propose a highly efficient DO-PSI protocol under the semi-honest model.

We conduct extensive experimental analyses in our protocol. Among existing PSI/PMT methods and their variants, we compare ours with FHE-based solutions [21, 45, 52] that can be naturally adapted to the DO-PSI setting, demonstrating its superior scalability in the number of data owners. Reflecting practical large-scale scenarios, we show that our protocol can process a billion-scale (2^{30} items) dataset distributed and outsourced over 1024 servers, i.e., 2^{20} items per server. For a membership test of 2048 items over outsourced sets, where all elements are 128-bit integers, ours achieves a total runtime of 67.6 s in a 10 Gbps bandwidth setting with a constant per-server communication cost of 10.48 MB. These are 4.26–11.16× and 14.36–38.20× improvements in the total runtime and communication overhead from previous solutions, respectively. For reproducibility, we publicly release our source code on GitHub.

Technical Overview. In DO-PSI, there are three types of entities: data owners, servers, and clients. Data owners encrypt and outsource their sets to designated servers, and remain offline thereafter. A client then queries the servers to learn the intersection between the query set and the union of outsourced sets, without revealing non-matching items. The client interacts with the servers only.

Existing methods for PSI are not suitable for the DO-PSI setting, and several useful building blocks for traditional PSI are not applicable *as is*. For example, evaluating a polynomial $f(x) = \prod_{x \in X}(x-y)$ has been widely used for instantiating PSI protocols [16, 21, 30, 85]. However, when checking $y \in \cup_{i=1}^{l} \mathcal{X}_i$ for distributed sets $\mathcal{X}_1, \ldots, \mathcal{X}_l$ held by each server, one should compute $\prod_{i=1}^{l}(\prod_{x \in \mathcal{X}_i}(x-y))$ to aggregate the membership results from servers, resulting in l multiplications. Instead, we exploit the equality circuit based on the value annihilating function (VAF) [45], which returns 0 for non-zero inputs and 1 otherwise, thereby enabling equality checks by applying it to the difference of two items (*i.e.*, $\mathsf{VAF}(x-y) = 1$ iff $x = y$). We can check $y \in \mathcal{X}$ by summing up $\sum_{x \in \mathcal{X}} \mathsf{VAF}(x-y) \neq 0$ or not. More importantly, even for distributed sets $\mathcal{X}_1, \ldots, \mathcal{X}_l$, we can aggregate membership results from servers via addition only, thus ensuring high scalability for the number of servers.

Efficiently evaluating the VAF is the key challenge in our framework. A natural choice in finite fields is the Fermat's little theorem-based mapping $x \mapsto 1 - x^{p^k-1}$ for $x \in \mathbb{F}_{p^k}$, which has been explored in prior studies on homomorphic equality circuits [38, 44]. However, it requires wider FHE plaintext slots during computation, degrading the efficiency. We avoid this by designing an efficiently computable mapping, called NPM, that maps a given long set item in $\mathbb{F}_{p^k}$ into a smaller field element in $\mathbb{F}_p$ while preserving nonzero property, i.e., a non-zero input always maps to a non-zero element while zero maps to zero. This allows us to design a VAF over $\mathbb{F}_{p^k}$ from one over a smaller field $\mathbb{F}_p$, thus removing the reliance on wider slots.

A naïve approach to designing an NPM is viewing $\mathbb{F}_{p^k}$ as a k-dimensional $\mathbb{F}_p$-vector space and applying a random inner product, resembling random projection-based dimensionality reduction [40]. However, such randomized mappings incur false positives, where a non-zero value maps to 0, with a probability of p^{-1}, which is non-negligible in typical FHE parameters. While repetition may reduce this error, it prevents compression of the input into a single element. Instead, we propose a deterministic, exact construction inspired by the Pell equation. Specifically, we employ the bivariate polynomial $f(x, y) = x^2 - dy^2$ for a non-quadratic residue d over $\mathbb{F}_p$, which provably satisfy the NPM property without false positives. Furthermore, we propose a recursive composition method for such f, obtaining a general NPM of k variables that is both exact and efficiently computable under homomorphic evaluation.

By combining our NPM with a VAF over $\mathbb{F}_p$, we design an efficiently evaluable VAF for $\mathbb{F}_{p^k}$. Finally, with a hashing technique tailored to our NPM-based VAF and a blinding technique to hide the cardinality of matched items across the outsourced sets, we obtain our DO-PSI protocol secure under the semi-honest model.

2 Related Works

(Multi-Party) Private Set Intersection. PSI is a problem where the client wants to privately learn the membership of multiple items in a server's set. Usual PSI protocols are designed for the two-party scenario, and each party can access its set items in plaintext. Several proposals have been made from various approaches, including OT [63, 64, 76], FHE [15, 16, 21, 52, 74, 83], or OKVS [10, 31, 33, 66]. Extending the functionality of PSI has also been studied, such as labeled-PSI [15, 21, 52] or circuit-PSI [35, 69, 74].

There are multi-party versions of PSI (mPSI) where the receiver learns the intersection of sets from multiple parties. Although several protocols have been proposed [14, 31, 46, 56, 84], these methods are not applicable for DO-PSI *as is*. This is because, in the DO-PSI setting, the receiver wants to learn which items in the query set belong to the union of sets from each party, not an intersection. On the other hand, there is another variant of mPSI, called quorum PSI [9, 14, 85], which fits our problem in terms of desired functionality. In quorum PSI, the receiver learns whether the query item is contained in the datasets of at least a threshold number, say τ, of parties. However, these existing methods have limited scalability for thousands or more parties [9, 14, 46, 84], or do not support encrypted datasets due to their inherent protocol design [31, 85].

Delegated Private Set Intersection. Delegated PSI (D-PSI) is a variant of PSI where each party delegates its encrypted data to a cloud server. The cloud server performs all the heavy computations for the set intersection. Several protocols have been proposed, e.g., based on (additive) HE [2, 43, 86], hashing techniques [3, 87], or (garbled) bloom filter [1, 26]. D-PSI can be extended to multi-party settings (D-mPSI) similarly to mPSI [1, 87], where multiple parties outsource encrypted data to a single cloud server. However, these protocols have limited scalability as they require the central party (cloud server) to manage and process all the storage and computations. Furthermore, they often require each data owner to stay online during the protocol execution [1, 3, 26, 87].

Private Membership Test. PMT is a special case of PSI where the client holds only a single set element. Several real-world applications based on the PMT have been proposed, e.g., malware detection [67, 77] or harmful media detection [47]. Recently, Vos et al. [81] proposed a privacy-preserving query protocol that enables a centralized entity to efficiently query sets held by multiple parties. Their protocol is suited for cross-silo federations, where privacy regulations and data fragmentation prevent direct data sharing. However, applying theirs to the DO-PSI scenario is not straightforward because their protocol design allows the centralized entity to learn the query. On the other hand, Koirala et al. [45] proposed a PMT protocol over encrypted and segmented sets, aiming to reduce the cost of aggregating results from many servers while ensuring the privacy of outsourced data. To this end, they constructed an approximated VAF over real numbers, which enables result aggregation via homomorphic summation only, making their protocol highly scalable to the number of data owners. However, their protocol suffers from huge communication and computation overheads from evaluating the VAF, and the set items' lengths are limited to at most 25 bits, resulting in low data precision and, thus, high false positive rates. In addition, they assumed that all the server's sets are disjoint from each other, which is unrealistic in practice.

Other Similar Techniques. Methods such as private information retrieval (PIR), oblivious RAM (ORAM), and trusted execution environments (TEEs) have been widely used for privacy-preserving

Table 1: Comparison of our protocol for DO-PSI with existing PSI/PMT works in terms of supporting "distributed" data sources, "outsourcing" in an encrypted form, and "scalability" wrt. the number of data owners. "Negl. FP" denotes that the false positive rate is below 2^{-40}.

Methods	Distributed	Outsourced	Scalability	Negl. FP
PMT/PSI [21, 52]	×	×	–	✓
mPSI [31, 84]	✓	×	×	✓
D-PSI [2, 26]	×	✓†	–	✓
D-mPSI [1, 87]	✓	✓†	×	✓
PSMT [45]	✓★	✓	△★	×
Ours	✓	✓	✓	✓

★This work requires the sets held by data owners to be disjoint and support sets of limited sizes only (e.g., at most 2^{25}).

†These works require the data owner to stay online during the protocol.

querying a database [49, 55, 77, 79]. However, PIR and ORAM focus on hiding the access pattern of the receiver to the sender's database. In PIR, the sender's dataset is typically assumed to be available in public, and dealing encrypted database is impossible or incurs heavy overhead. Although ORAM supports encrypted databases, they do not scale well to a large number of parties [79]. TEEs have been repeatedly shown to suffer from side-channel attacks, making their use skeptical for highly sensitive use cases [59, 72, 82].

We compare representative prior works with ours in Table 1.

3 Preliminaries

Notation. For $k \in \mathbb{N}$, we denote $[k] = \{1, \ldots, k\}$. We denote log as a logarithm with base 2. For a prime number p and a positive integer k, we denote $\mathbb{F}_{p^k}$ as a finite field of order p^k. For a ring R, we denote R^* as a collection of elements in R excluding zero divisors. We denote $R[X]$ as a polynomial ring of an indeterminate X with coefficients in R. For a polynomial $f(X) \in R[X]$, we denote the quotient ring of $R[X]$ over $f(X)$ as $R[X]/\langle f(X)\rangle$. For a finite set $\mathcal{S}$, we denote $x \xleftarrow{\$} \mathcal{S}$ as a uniform random sampling over $\mathcal{S}$.

Fully Homomorphic Encryption (FHE). FHE, first introduced by Gentry [34], is a cryptographic primitive that enables the execution of an arithmetic circuit in an encrypted domain. Several FHE schemes have been proposed, including B/FV [28], BGV [13], and CKKS [17]. The security of them is guaranteed by the hardness of the ring learning-with-errors (RLWE) problem [50]. The homomorphic operations of these schemes are defined over the polynomial ring $\mathbb{F}_p[X]/\langle\Phi_M(X)\rangle$ with (polynomial) addition and multiplication, where $\Phi_M(X) = X^N + 1$ is the M-th cyclotomic polynomial for $M = 2N$ and M is the power of two. Note that usual FHE schemes are *leveled*, i.e., the number of multiplications is limited to some number because of the growth of noise in RLWE samples. In our protocol, we consider a leveled version of these schemes.

Many FHE schemes support SIMD (Single Instruction, Multiple Data) operations to maximize the advantage of parallel computation. To this end, they utilize NTT (number-theoretic transformation) to encode a vector in $\prod_{i=1}^{N} \mathbb{F}_p$ into a polynomial $\mathbb{F}_p[X]/\langle X^N + 1\rangle$, so that homomorphic operations on $\mathbb{F}_p[X]/\langle X^N + 1\rangle$ correspond to component-wise operations in $\prod_{i=1}^{N} \mathbb{F}_p$. In addition, by utilizing the structure of NTT evaluation points, one can implement more advanced operations in $\prod_{i=1}^{N} \mathbb{F}_p$, e.g, rotating components or evaluating Frobenius automorphism for each component, by modifying the encoded polynomial in $\mathbb{F}_p[X]/\langle X^N + 1\rangle$.

Threshold FHE (ThFHE). ThFHE is a thresholdized FHE that supports splitting a FHE secret key via secret shares and broadcasting them to several parties, say l. Later, the decryption can be done by combining *partial* decryptions of more than a pre-determined number, say α. Each partial decryption can be computed from a secret key share. We define the (α, l)-threshold FHE as a tuple of PPT algorithms ThFHE = (Setup, Enc, Eval, PartialDec, Combine), whose functionalities are as follows:

- $\mathsf{Setup}(1^\lambda, 1^D, params) \to (pk, evk, \{sk\}_{i=1}^{l})$: It takes a security parameter λ, the maximum depth D, and the FHE parameters $params$ as inputs, returning a public key pk, evaluation key evk, and a l-secret share $\{sk\}_{i=1}^{l}$ of the secret key.
- $\mathsf{Enc}(pk, msg) \to ct$: It takes a public key pk and a plaintext msg as inputs, returning a ciphertext ct, an encryption of msg.
- $\mathsf{Eval}(evk, \{ct_i\}_{i\in[k]}, C) \to \widehat{ct}$: It takes an evaluation key evk, k ciphertexts $\{ct_i\}_{i=1}^{k}$, and a circuit C with k inputs as inputs, returning a ciphertext $\widehat{ct}$ corresponding to $C(v_1, \ldots, v_k)$, where v_i is a plaintext corresponding to ct_i for $i \in [k]$. It can also take plaintext inputs when C contains scalar operations.
- $\mathsf{PartialDec}(sk_i, ct) \to z_i$: It takes a partial secret key sk_i and a ciphertext ct as inputs, returning a partial decryption result z_i.
- $\mathsf{Combine}(pk, \{z_i\}_{i\in[\alpha]}) \to m \cup \{\perp\}$: It takes a public key pk and α partial decryption results z_i from partially decrypting the same ciphertext with different partial secret keys, returning a plaintext m or the symbol $\perp$ that stands for decryption failure.

The Setup algorithm can be implemented by either secure multiparty computation (SMPC) protocols [6, 54] or the setup and distribution by the trusted party [11, 41]. Note that the latter can be aided by a trusted hardware such as TEE.

Semi-Honest Adversary Model. Let π be an protocol with l parties $\mathcal{P} = \{P_1, \ldots, P_l\}$ for a functionality $\mathcal{F} = (\mathcal{F}_i)_{i\in[l]}$, where each party P_i receives $\mathcal{F}_i(x_1, \ldots, x_l)$ on its input x_i. We say that π securely implements $\mathcal{F}$ against the semi-honest adversaries if there is a PPT algorithm $\mathfrak{S}$ that simulates the views of corrupted parties from their inputs, corresponding outputs, and a security parameter. The views of parties include their inputs, all messages they received, and internal randomness used during the protocol execution. We denote the view of party P_i as $\mathcal{V}_i^{\pi}(x_1, \ldots, x_l, \lambda)$, and the output of all parties as $\pi(x_1, \ldots, x_l, \lambda)$.

Definition 1. *Let $\mathcal{F} = (\mathcal{F}_i)_{i\in[l]}$ be a functionality and π be an l-party protocol. We say that the π securely implements the functionality $\mathcal{F}$ in the presence of semi-honest adversaries $\mathcal{A}$ that can corrupt any subset of parties $\mathcal{P}^* \subset \mathcal{P}$ with $|\mathcal{P}^*| \le \alpha - 1$ if there exists PPT algorithm $\mathfrak{S}$ such that for a security parameter $\lambda \in \mathbb{N}$,*

$$\{\mathfrak{S}(1^\lambda, \{x_i, \mathcal{F}_i(x_1, \ldots, x_l)\}_{P_i\in\mathcal{P}^*}), \mathcal{F}(x_1, \ldots, x_l)\}_{x_1,\ldots,x_l,\lambda}$$
$$\overset{c}{\equiv} \{\{\mathcal{V}(x_1, \ldots, x_l, \lambda)_i^{\pi}\}_{P_i\in\mathcal{P}^*}, \pi(x_1, \ldots, x_l, \lambda)\}_{x_1,\ldots,x_l,\lambda},$$

where x_i's are inputs of the protocol and $\overset{c}{\equiv}$ denotes computational indistinguishability between the two distributions.

4 PSI over Distributed and Outsourced Data

We formalize *Distributed and Outsourced Private Set Intersection (DO-PSI)* to capture deployments with many data owners and outsourced, end-to-end–encrypted storage and compute. We first construct

Parameters: Let $\mathcal{S}_1, \ldots, \mathcal{S}_{l-1}$ denote the servers and C the client, comprising a total of l parties. We denote the dataset outsourced to each S_i as $\mathcal{X}_i$ for $i \in [l-1]$. We denote $\mathcal{U}$ as the universe set where each set element lies.

Inputs:

- C: A query set $\mathcal{Y} \subset \mathcal{U}$.
- $\mathcal{S}_i$: An encryption of a set $\mathcal{X}_i \subset \mathcal{U}$.

Outputs:

- C: The intersection $\mathcal{Y} \cap (\bigcup_{i=1}^{l-1} \mathcal{X}_i)$.
- $\mathcal{S}_i$: Nothing.

Figure 1: Ideal functionality $\mathcal{F}_{\text{DOPSI}}$ of DO-PSI

DO-PSI for a singleton-client regime, i.e., PSI with a one-element client set, then lift the construction to larger sets via hashing. Our generic framework builds on threshold FHE and enables secure, communication-efficient aggregation of per-server membership outcomes, while avoiding any centralized plaintext holder.

4.1 Problem Formulation

DO-PSI extends PSI to settings with distributed, outsourced, and encrypted sets. In DO-PSI, there are three types of entities:

- **Data Owner** ($\mathcal{DO}_i$) handles a set $\mathcal{X}_i$ who encrypts and outsources encrypted $\mathcal{X}_i$ to a server. They could be healthcare service providers, business owners, or governmental departments.
- **Server** ($\mathcal{S}_i$) handles an encrypted set of $\mathcal{X}_i$ that was outsourced from the data owner $\mathcal{DO}_i$. Servers can be cloud platforms, HPC clusters, or secure compute nodes.
- **Client** (C) handles a query set Y and wants to retrieve $Y \cap (\cup_i \mathcal{X}_i)$ without revealing non-matching items to any of the servers.

Each data owner encrypts and outsources its dataset to an untrusted server, and remains offline, playing no role during the protocol execution. This mirrors delegated PSI and eliminates the need for local storage or continuous availability for the data owner [42]. The server possesses ample storage resources and computing power to operate on encrypted data. Note that several data owners can outsource their datasets to the same server under the same encryption key. To simulate a setting requiring high scalability, we assume that each data owner outsources its data to a dedicated server.

The goal of DO-PSI is for the client to securely learn the intersection between the client's set Y and the union $\bigcup_{i=1}^{l-1} \mathcal{X}_i$ of sets from data owners leaking any information about the outsourced data or the query, e.g., which server the intersection is coming from. To ensure the data privacy of each $\mathcal{DO}_i$ for the set $\mathcal{X}_i$, we assume that each $\mathcal{X}_i$ is encrypted under some common public encryption key and each server alone lacks the ability to decrypt. Since each data owner can process their data in advance to remove duplicate items, without loss of generality, we assume that each $\mathcal{X}_i$ has no duplicates internally. Unlike existing works [45], we allow common items among different data owners' sets $\mathcal{X}_i$ and $\mathcal{X}_j$. We describe the ideal functionality $\mathcal{F}_{\text{DOPSI}}$ of the DO-PSI in Fig. 1.

Adversary & Security Model. We consider the semi-honest model, where all parties honestly follow the protocol but attempt to learn as much private information as possible during its execution. Since each data owner will not engage in the protocol after outsourcing, we only consider the collusion between the servers and the client. We implement $\mathcal{F}_{\text{DOPSI}}$ using (α, l)-threshold FHE. The secret key shares are managed by the servers and the client, assuming that the number of corruptions between them is at most $(\alpha - 1)$. Each data owner's data is secured using the threshold FHE. Remark that we let the client C hold the secret key share to defend against an external adversary who attempts to reconstruct the membership results from the eavesdropped partial decryptions. Although such an adversary was not considered in our ideal functionality as the intersection result *per se* could be private information.

4.2 A Generic and Scalable Solution for DO-PSI

We present a generic framework to solve the DO-PSI problem, which is highly scalable to the number of data owners. We temporarily assume a client set size, $|\mathcal{Y}| = 1$ with a single query element y. This assumption will be removed in Section 5.3 to support arbitrary client set sizes by using hash tables.

Let K be a field where each set item belongs to and $\text{eq} : K \times K \to \{0, 1\}$ be the equality operator over K such that $\text{eq}(x, y) = 1$ if $x = y$, 0 otherwise. Using this, one can securely check whether an element $y \in K$ belongs to the set $\mathcal{X} \subset K$ by following procedure:

(1) For each $x \in \mathcal{X}$, securely compute $s_x \leftarrow \text{eq}(x, y)$.
(2) Compute $\bar{s} \leftarrow \sum_{x \in \mathcal{X}} s_x$ and check $\bar{s} = 1$ or not.

The main advantage of this approach is that aggregating each output s_x can be done via additions only, as opposed to several HE-based PSIs that require expensive multiplications [15, 16, 21]. In particular, the summation $\bar{s}$ can be further additively aggregated across different servers having different sets $\mathcal{X}_i$ to check the membership of y in $\bigcup \mathcal{X}_i$, ensuring high scalability for the number of servers.

In the design of the equality operator, we are inspired by the *value annihilating function* (VAF) $f : K \to \{0, 1\}$, which returns nonzero values given 0s as the inputs and 0s for nonzero inputs [45]. With this function, we can implement $\text{eq}(x, y) = f(x - y)$. Unlike the previous work that considered real-valued inputs only, we extend the definition over an arbitrary field K as follows:

Definition 2 (Value Annihilating Function). *Let K be a field. Then a function $f : K \to \{0, 1\}$ is called a value-annihilating function (VAF) if the following property holds.*

$$f(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise.} \end{cases}$$

From this, we can construct a protocol to solve the DO-PSI problem using the (α, l)−ThFHE that supports operations over the field K. After running ThFHE.Setup between the client and servers and outsourcing the encrypted set $ct_{\mathcal{X}_i}$ from each data owner, the protocol is executed in the following manner. First, the client encrypts the query element $y \in K$ using ThFHE.Enc and broadcasts the resulting ciphertext ct_y to all the servers. Then, each server homomorphically evaluates the VAF on the difference of ciphertexts $ct_{\mathcal{X}_i}$ and ct_y, and additively aggregates the resulting ciphertexts.

To aggregate the result ciphertexts from different parties, we assume that there is a *leader* server that facilitates aggregating individual VAF evaluations from each server. Without loss of generality, we assume that the first server, $\mathcal{S}_1$, plays the role of the leader. Each server sends the locally aggregated ciphertext to the leader $\mathcal{S}_1$, and $\mathcal{S}_1$ additively aggregates them to obtain the final aggregation ciphertext $\bar{s}$. To facilitate the decryption of the result, the leader

Parameters: l parties $\mathcal{S}_1, \ldots, \mathcal{S}_{l-1}$ and C, each of which corresponds to the servers and the client, respectively. λ is the security parameter. A VAF f and an arithmetic circuit $C_f(x, y) = f(x - y)$. A circuit C_Σ for summing up all the given inputs. The universe set K as a field.

Setup:

(1) All parties jointly run the protocol ThFHE.Setup using the security parameter λ, the FHE depth parameter D, and other FHE parameters $params$ as agreed inputs. Each party $\mathcal{S}_i$ learns the encryption key pk, evaluation key evk, and the secret share sk_i of the secret key. Client C learns sk_l.
(2) Each server $\mathcal{S}_i$ sends pk to the corresponding data owner $\mathcal{DO}_i$ having the database $\mathcal{X}_i$, receiving the encryption of a dataset $\mathcal{X}_i$, $ct_{\mathcal{X}_i} := \{\mathsf{ThFHE.Enc}(pk, x) : x \in \mathcal{X}_i\}$.

Protocol:

(1) **Broadcast**: C sends $ct_y \leftarrow \mathsf{ThFHE.Enc}(pk, y)$ to each $\mathcal{S}_i$.
(2) **Evaluate VAF**: For each $\mathcal{S}_i$ and $ct_x \in ct_{\mathcal{X}_i}$, homomorphically subtract the query ciphertext from the outsourced ciphertext and evaluate the VAF by running $\mathsf{ThFHE.Eval}(evk, \{ct_x, ct_y\}, C_f) \rightarrow s_x^i$. After locally aggregating all s_x^i to obtain $\mathsf{ThFHE.Eval}(evk, \{s_x^i, C_\Sigma\}) \rightarrow \bar{s}_i$, send $\bar{s}_i$ to the leader server $\mathcal{S}_1$.
(3) **Aggregate**: $\mathcal{S}_1$ additively aggregates all the result by running $\mathsf{ThFHE.Eval}(evk, \{\bar{s}_i\}_{i\in[l]}, C_\Sigma) \rightarrow \bar{s}$. Then $\mathcal{S}_1$ broadcasts the result to C and $(\alpha - 1)$ servers.
(4) **Partial Decryption**: Each party having the aggregation result runs $\mathsf{ThFHE.PartialDec}(sk_i, \bar{s}) \rightarrow z_i$ and sends the result to C.
(5) **Result Retrieval**: C finally obtain the intersection result by running ThFHE.Combine on received partial decryptions, obtaining m. C confirms y is in the dataset pool if $m \neq 0$, otherwise y is not present in the dataset pool.

Figure 2: Basic protocol for solving DO-PSI

broadcasts the final ciphertext to randomly selected $(\alpha - 1)$ servers and the client. These servers run the partial decryption using their share of the secret key, and then forward the partial decryption result to the client. Finally, the client learns the result by running the Combine algorithm; if there is no matching, then the result should be 0, otherwise it should be a nonzero field element. We describe the basic protocol based on this approach for a single query item in Fig. 2. Later, we will build our full protocol for an arbitrary client set size based on this approach.

Key Aspects when Instantiating the Protocol. When instantiating the DO-PSI protocol using the above-mentioned approach, the design of a VAF plays a significant role in its efficiency. Existing protocol [45] that follows this approach uses VAF for $K = \mathbb{R}$ through polynomial approximation techniques. However, such a setting necessitates a highly accurate approximation in $\mathbb{R}$ to ensure low false positives when supporting large set items, which results in huge communication and computation overheads.

In addition, we note that the value in the aggregated ciphertext $\bar{s}$ might contain the cardinality of the matched items across the servers' data. However, such leakage is not permitted according to the ideal functionality of $\mathcal{F}_{\text{DOPSI}}$. One naïve mitigation, as in [45], is to employ an assumption that all the data owners must have disjoint items among them. However, such an assumption about the data is rather too strong and unrealistic in practical settings as each data owner does not share their sensitive data with each other and is not aware of any potential duplicate items.

5 Our Solution

We now present our key techniques to efficiently instantiate the framework in Fig. 2. We first introduce VAFs over the finite fields, which do not require polynomial approximation. Among them, we propose a novel technique called *nonzero-preserving mapping*, which sharply reduces the computational cost for the VAF evaluation. Next, we lift the construction to support large query sets via hashing and propose several optimizations that significantly reduce both computation and communication costs. Finally, we employ a technique to randomly blind the aggregated results by the leader server, even in the existence of collusion between parties. With these tools, we finally construct a protocol that securely implements the functionality $\mathcal{F}_{\text{DOPSI}}$ under the semi-honest model.

5.1 VAFs from Fermat's Little Theorem

To avoid the efficiency and accuracy issues incurred by the polynomial approximation over $\mathbb{R}$, we shift our attention to designing a VAF over a finite field $K = \mathbb{F}_{p^k}$. Here, we can straightforwardly design a VAF from Fermat's little theorem over finite fields, which tells us that $(\mathbb{F}_{p^k})^*$ is a multiplicative cyclic group of order $p^k - 1$. That is, $x^{p^k-1} = 1$ for all $x \in (\mathbb{F}_{p^k})^*$. Therefore, the function $f(x) = 1 - x^{p^k-1}$ satisfies that $f(x) = 1$ if and only if $x = 0$; otherwise $f(x) = 0$, so $f(x)$ is indeed a VAF defined over $\mathbb{F}_{p^k}$. Implementing operations over $\mathbb{F}_{p^k}$ is possible by utilizing the well-known technique based on the Chinese remainder theorem (CRT) [73].

However, dealing with elements in $\mathbb{F}_{p^k}$ requires wider slots in FHE to represent them, which can lower the throughput. In addition, the choices of k and N are restricted because of the CRT technique, resulting in limited flexibility for the supported size of set elements. Here, one can exploit the idea of Kim et al. [44]: view $\mathbb{F}_{p^k}$ as a $\mathbb{F}_p$-vector space $\prod_{i=1}^k \mathbb{F}_p$ and then compute $(x_1, \ldots, x_k) \in \prod_{i=1}^k \mathbb{F}_p \mapsto \prod_{i=1}^k (1 - x_i^{p-1})$. Although this mapping can be evaluated with the same number of multiplications and depths as the previous VAF, their method still requires wider slots to represent each set item for a single ciphertext, reducing packing density and thus throughput.

5.2 Designing VAFs over Finite Fields through Nonzero-Preserving Mapping

To overcome the limitations above, we introduce *nonzero-preserving mapping*, a novel method that enables us to construct a VAF over $\mathbb{F}_{p^k}$ with operations in $\mathbb{F}_p$ only, thereby avoiding wide-slot encodings and maintaining high packing density.

High-Level Idea. Imagine that there is a multivariate polynomial $\varphi : \prod_{i=1}^k \mathbb{F}_p \rightarrow \mathbb{F}_p$ satisfying $\varphi(x_1, \ldots, x_k) = 0$ if and only if $x_1 = \cdots = x_k = 0$. That is, for the VAF $f(x) = 1 - x^{p-1}$ for $\mathbb{F}_p$, we can observe that the composition $\Phi = f \circ \varphi$ becomes a VAF for $\prod_{i=1}^k \mathbb{F}_p$. This is because $\Phi(x) = 1$ if and only if $\varphi(x) = 0$, i.e, $x = (0, \ldots, 0) \in \prod_{i=1}^k \mathbb{F}_p$. We can view the role of φ as compressing the given vector to a single component while preserving its nonzero property, i.e., a nonzero input vector always gives the nonzero output. Thanks to this property, computing a VAF over a smaller field $\mathbb{F}_p$ is sufficient rather than computing it over the whole field $\mathbb{F}_{p^k}$ after evaluating φ. We call such a multivariate polynomial a nonzero-preserving mapping and formally define it below.

Definition 3 (Nonzero-Preserving Mapping). *Let K be a field and $f : \prod_{i=1}^{k} K \to K$ be a multivariate polynomial. Then f is called a nonzero-preserving mapping over K with k-variables (k-NPM over K) if $f(x_1, \ldots, x_k) = 0$ is equivalent to $x_1 = \cdots = x_k = 0$.*

A Naïve Approach: Random Inner Product. When we admit failure probability, i.e., the nonzero vector could be probabilistically mapped to 0 while the zero vector always is mapped to 0, there is a trivial way to construct an NPM through a random inner product. Precisely, for a vector $(x_1, \ldots, x_k) \in \prod_{i=1}^{k} \mathbb{F}_p$, sample $r_i \xleftarrow{\$} \mathbb{F}_p$ for $i \in [k]$ and compute $\sum_{i=1}^{k} x_i r_i$. This plays a similar role as an NPM because the random variable $x_i r_i$ follows the random uniform distribution over $\mathbb{F}_p$ if $x_i \neq 0$; otherwise, it becomes a constant distribution of 0. That is, $\Pr\left[\sum_{i=1}^{k} x_i r_i = 0 \mid \exists i \in [k] \text{ such that } x_i \neq 0\right] = \frac{1}{p}$.

However, this approach *per se* cannot be used for designing an NPM. To ensure a negligible failure probability, p should be sufficiently large, but this yields a significant overhead for evaluating the VAF over $\mathbb{F}_p$ after NPM. One may attempt to repeatedly run the random inner product and check whether the results from each iteration are all zero. But this does not fundamentally solve our problem, though it can reduce the dimension of the input vector akin to random projection [40]. Let $\xi_1, \ldots \xi_w \in \mathbb{F}_p$ as the w random inner product results to ensure the failure probability of $\frac{1}{p^w}$. Checking all ξ_i's are zero is equivalent to running a w-NPM on inputs $\xi_1, \ldots, \xi_m$. However, because of the failure probability, we cannot further reduce the number of variables in the same way. Therefore, we need an *exact* NPM without failure probability at the end.

Nonzero-Preserving Mapping from Pell Equation. Pell equation is an integer equation of the form $x^2 - dy^2 = 1$, where x, y are indeterminates and d is a positive integer. Its properties have been studied a lot, and its connections to the number fields are widely known. Particularly, if d is not a square, then $f(x, y) := x^2 - dy^2$ becomes a multiplicative norm on the extension field $\mathbb{Q}(\sqrt{d}) = \{x + \sqrt{d}y : (x, y) \in \mathbb{Q} \times \mathbb{Q}\}$, i.e., $f(x, y) = 0$ if and only if $x = y = 0$.

Such a property is also preserved when we consider the general field K, requiring that $X^2 - d$ is an irreducible polynomial over K. Therefore, we can regard the aforementioned bivariate polynomial as a 2-NPM defined over K, as stated in the following Theorem 1.

Theorem 1. *Let K be a field and $d \in K$ be an element s.t. $X^2 - d$ is irreducible over K. Then $f(x_1, x_2) = x_1^2 - dx_2^2$ is a 2-NPM over K.*

The key advantage of this choice is its simplicity: two homomorphic multiplications with one multiplicative depth are enough for evaluation. We use it as a building block for designing k-NPM.

Composition of NPMs. To support further long items over $\prod_{i=1}^{k} \mathbb{F}_p$, we should be able to construct k-NPMs for any $k \in \mathbb{N}$. One possible approach is to generalize the Pell equation, but this results in extremely huge computational overhead as k grows[1]. Instead, we construct new NPMs by composing the existing ones. The core idea is to view a 2-NPM as a *bridge* to compose other NPMs. More precisely, let us consider 2-NPM f_2, m-NPM f_m and n-NPM f_n over the same field K. Then we can construct a $(m+n)$-NPM f_{m+n} by,

$$f_{m+n}(x_1, \ldots, x_{m+n}) = f_2\Big(f_m(x_1, \ldots, x_m), f_n(x_{m+1}, \ldots, x_{m+n})\Big).$$

[1]We provide detailed discussions about generalized Pell equations in Appendix B.

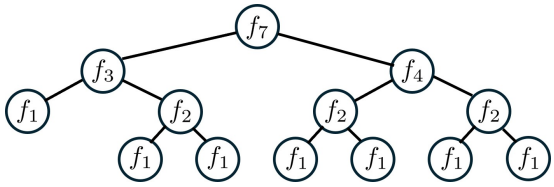


Figure 3: Example of the summation tree for computing f_7.

Checking f_{m+n} becomes an $(m+n)$-NPM is straightforward, because $f_{m+n}(x_1, \ldots, x_{m+n}) = 0$ if and only if $f_m(x_1, \ldots, x_m) = 0$ and $f_n(x_{m+1}, \ldots, x_{m+n}) = 0$ by definition. With this idea and a 2-NPM in Theorem 1, we can construct any k-NPM f_k.

Depth-Efficient Composition. There are several ways to compose f_k. For example, one can inductively compose 1-NPM, i.e., $f_{i+1}(x_1, \cdots x_i) = f_2\Big(f_i(x_1, \ldots, x_i), x_{i+1}\Big)$ for $i \in [k-1]$. This requires $k-1$ sequential evaluations of f_2, so the required depth is $k-1$. We show that there is a *good* composition where $\lceil \log k \rceil$ depth suffices while requiring the same number of evaluations of f_2.

First observe that a computational graph for calculating f_k can be viewed as a binary tree T with the following property:

- T has k leaf nodes, and each leaf node has a value of 1.
- Each parent node has a sum of the values in its children.

Each value in the node is the current number of variables in an NPM, and ascending to the ancestor is akin to composing two NPMs via f_2. We visualize a computational graph for calculating f_7 in Fig. 3.

We can observe that (1) the depth required for computation depends on the depth of T, and (2) the number of evaluations of f_2 depends on the number of non-leaf nodes. Since the number of non-leaf nodes is always $k-1$, minimizing the depth gives the optimal way to evaluate f_k. Hence, splitting k as *balanced* as possible is the best strategy. We summarize this approach in Theorem 2, whose proof is straightforward via mathematical induction.

Theorem 2. *Let K be a field and $d \in K$ be an element s.t. $X^2 - d$ is irreducible over K. For $k \in \mathbb{N}$, we define $f_k : \prod_{i=1}^{k} K \to K$ by*

- *$f_1(x_1) = x_1$ and $f_2(x_1, x_2) = x_1^2 - dx_2^2$.*
- *$f_k(x_1, \ldots, x_k) = f_2\Big(f_{\lfloor \frac{k}{2} \rfloor}(x_1, \ldots, x_{\lfloor \frac{k}{2} \rfloor}), f_{\lceil \frac{k}{2} \rceil}(x_{\lfloor \frac{k}{2} \rfloor + 1}, \ldots, x_k)\Big)$.*

Then f_k is a k-NPM over K.

Computational Cost. Evaluating f_k requires $(k-1)$ evaluation of f_2 with consuming $\lceil \log k \rceil$ depths. Since each evaluation of f_2 requires 2 multiplications, the total number of multiplications is $2(k-1)$. To make it a VAF, we need to compose a VAF over K after evaluating f_k. If $K = \mathbb{F}_p$, then the total number of multiplications and depth consumption are $2(k-1) + \lceil \log p \rceil$ and $\lceil \log k \rceil + \lceil \log p \rceil$, respectively. Unlike the VAF discussed in Section 5.1 and the NPM-free method of Kim et al. [44], NPM-based methods do not require multiple slots in a single ciphertext to represent each item. Thus, the NPM-based method has better amortized cost than the former one, though it requires more computation for one evaluation.

Probabilistic NPM. We also note that the above exact NPM can finally be combined with the naïve random inner product idea. More precisely, if p^{-w} is sufficiently small for some $w < k$, then we can first apply w random inner products for the given inputs and apply the w-NPM from the above construction. Since these inner

products consist of additions and scalar multiplications, we can reduce both the number of non-scalar multiplications and depth consumption. We will denote such an NPM as a probabilistic NPM.

5.3 PSI for Large Query Sets via Hashing

We show how to integrate hashing (bucketing) with DO-PSI, which is a standard technique to convert a PMT into PSI [15, 16, 21, 52, 63].

Vector-friendly Hashing Table. When applying the hashing technique with our NPM-based VAF approach, since we are dealing with k-dimensional $\mathbb{F}_p$-vector, each component of the set elements should be *well-positioned* in the hash table. More precisely, we need to ensure that (1) components from the same vector are placed in the same column of the table, and (2) components of the same position are aligned for comparison during the PSI for a single item.

To address these, we design a hash table for vectors that forces components from the same vector to lie in the same column of the table. Our strategy is to segment the hash table of N bins for each component and assign it to the same position for each sub-table. To be precise, let us consider a hash function $h : \prod_{i=1}^{k} \mathbb{F}_p \to [m]$. We assume that $N = km$ for simplicity. From this, we define a function $H : \prod_{i=1}^{k} \mathbb{F}_p \to \{S \subset [N] : |S| = k\}$ that assigns the position of each component of the input vector, as follows:

$$H(x_1, \ldots, x_k) = \{i \cdot m + j : i \in [k] \text{ and } j = h(x_1, \ldots, x_k)\}.$$

Here, for $i \in [k]$, x_i is placed in $(i \cdot m + j)^{\text{th}}$ bin of the table.

By construction, $H(\vec{x}) \cap H(\vec{y}) = \emptyset$ whenever $h(\vec{x}) \neq h(\vec{y})$, otherwise these two sets are equal. In addition, each component of the input vector is placed in the same position as the sub-tables. Due to these properties, we can check that the resulting construction satisfies both aforementioned conditions. Note that we can utilize any hashing technique because we use the positioning function h as a black-box. For example, we can use cuckoo hashing [60], which has been widely used in many FHE-based PSI constructions [15, 16, 21]. In addition, to prevent leakages from the number of columns in the hash table, we pad each bin with dummy values to ensure that the number of items per bin is always a pre-defined public number B.

Query Extraction from Hashing Table Structure. In our hashing table H, the i^{th} component of each input vector lies in bins from $i \cdot m$ to $(i+1) \cdot m - 1$. To leverage this structure, we first extract i^{th} components by multiplying a masking vector $\sum_{j=1}^{m} \vec{e}_{i \cdot m + j}$, where $\vec{e}_i$ is the one-hot vector whose i^{th} component is 1, otherwise 0. We then replicate the extracted component $k = N/m$ times to fill the whole ciphertext slots by rotation and addition. This gives k ciphertexts in total, one for each position of the input vectors, thus enabling our NPM and VAF techniques. In addition, since there are k duplicates for each ciphertext, we can process k columns of each server's hashing table at once by packing them in a single ciphertext. This allows for k-fold speedup for computing the intersection. We visualize the overall process in Fig. 4.

Aggregating Intersection Results. After evaluating the NPM and VAF, the server can aggregate the local intersection results by summing them up from each k column, obtaining a single ciphertext. Within this ciphertext, the slots from $i \cdot m$ to $(i+1) \cdot m - 1$ contain intersection results for the input vector and i^{th} column of all sub-tables. Hence, by summing up these k partitioned slots, i.e., rotation and addition of stride N/k, the server obtains a single ciphertext that contains the result for the server's set. Finally, the leader server can aggregate the results from the others by simply summing them.

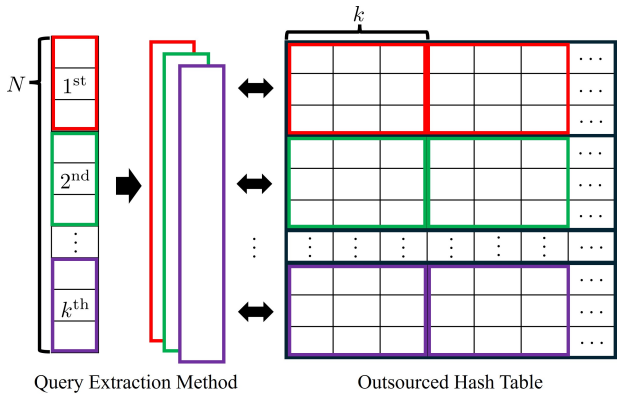


Figure 4: Our vector-friendly hashing and query extraction method to handle k columns at once. Best viewed in color.

5.4 Hiding the Cardinality of Matched Items

The aggregated ciphertext as above may leak the cardinality of matched items across the whole set, which is not permitted in DO-PSI. To prevent this, we must randomly blind the aggregated ciphertext while still allowing checking (non-)intersection. One naïve solution is to let the leader server sample a plaintext that encodes a vector in $\prod_{i=1}^{N} \mathbb{F}_p^*$ and multiply it to the aggregated ciphertext. This completely hides the cardinality while enabling the client to check the membership of each item, i.e., the resulting ciphertext contains 0 if and only if the item is not in the set. However, because a single party (the leader server) knows the whole blinding factor, the receiver can retrieve the cardinality information through collusion.

To mitigate this, we let all servers contribute to the blinding factor. For each server $\mathcal{S}_i, i \in [l-1]$, it samples a random vector $\vec{r}_i$ in $\prod_{i=1}^{N} \mathbb{F}_p$. The servers then send their encryption to the leader server along with the VAF result. The leader server homomorphically computes $\bar{r} = \sum_{i=1}^{l-1} \vec{r}_i$ and considers it as the blind factor for the aggregated VAF result, $\bar{z}$. At the end, the receiver obtains $\bar{r} \odot \bar{z}$ for the component-wise multiplication $\odot$, and can decide the membership of the query by checking whether all received results are zero or not. Since each component of $\bar{r}$ is independent and follows the uniform distribution over $\mathbb{F}_p$, the resulting value may be all zeroes even when there are matched items. However, this probability can be made negligible. Due to our aggregation strategy, the intersection result for each query vector is duplicated over k positions. Hence, if there were an intersection, then the probability of all the values in the duplicated position becoming 0 is at most p^{-k}. Thus, the receiver cannot learn the cardinality unless it colludes with all the servers because all the servers contribute to the blind factor $\bar{r}$.

5.5 The Proposed Protocol

By leveraging all these techniques, we implement the DO-PSI protocol following the framework in Fig. 2. For completeness, we provide the detailed procedure of the protocol. The full protocol is described in Fig. 5. Due to space constraints, we provide detailed algorithms for NPM and the query compression technique in Appendix C.

Parameters: Servers $\mathcal{S}_1, \ldots, \mathcal{S}_{l-1}$ and a client C. C holds a set $\mathcal{Y}$. FHE params $params$, depth D, security parameter λ, the number of variables in NPM k, and the vector-friendly hashing H. Each set element belongs to $[0, 2^w)$ and for a plaintext modulus p of a FHE, $2^w < p^k$ holds. Arithmetic circuits C_ϕ, C_{VAF}, C_Σ, C_{RotAdd} corresponding to NPM, VAF, summation, and rotation-and-addition technique, respectively.

Setup and Preparing the Encrypted Data:

- All parties jointly run $\mathsf{ThFHE.Setup}(1^\lambda, D, parmas) \to (pk, evk, \{sk_i\}_{i=1}^l)$. Each $\mathcal{S}_i$ obtains a secret key share sk_i and C learns sk_l.
- Each $\mathcal{S}_i$ sends pk to the corresponding data owner $\mathcal{DO}_i$, which holds a set $\mathcal{X}_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,D_i}\}$. Then each $\mathcal{DO}_i$ do the following:
  (1) For each $x_{i,j}$ for $j \in [D_i]$, parse $x_{i,j}$ into $\vec{x}_{i,j} := (x_{i,j,1}, \ldots, x_{i,j,k}) \in \prod_{i=1}^k [0, p)$ such that $x_{i,j} = \sum_{m=1}^k x_{i,j,m} \cdot p^{m-1}$.
  (2) Build a hash table $T_i : [N] \times [B_i] \to \mathbb{F}_p \cup \{\perp\}$ from H and $\{\vec{x}_{i,j}\}_{j \in [D_i]}$, where B_i denotes the maximum number of items per bin. Vacant slots are filled with a dummy value $\perp$.
  (3) For $Q_i := \lceil \frac{B_i}{k} \rceil$, compute $ct_{q,m} \leftarrow \mathsf{ThFHE.Enc}(pk, \vec{x}_{q,m})$, where $\vec{x}_{q,m}$ reads out the sub-table of T_i from $(\lceil N/k \rceil \cdot (m-1)+1)^{\text{th}}$ to $(\lceil N/k \rceil \cdot m)^{\text{th}}$ rows and from $(k \cdot (q-1)+1)^{\text{th}}$ to $(k \cdot q)^{\text{th}}$ in a column-major order. When $k \cdot q^* + m^* > B_i$ for some q^* and m^*, we set such a slot to $\perp$.
  (4) Send the resulting ciphertexts $ct_{\mathcal{X}_i} := \{ct_{q,m} : q \in [Q_i], m \in [k]\}$ to the corresponding server $\mathcal{S}_i$.

Client-Side Query Processing:

(1) For each $y_i \in \mathcal{Y}$, parse y_i into $\vec{y}_i := (y_{i,1}, \ldots, y_{i,k}) \in \prod_{m=1}^k [0, p)$ such that $y = \sum_{m=1}^k y_m \cdot p^{m-1}$.
(2) Build a hash table $T_l : [N] \times [B_l] \to \mathbb{F}_p \cup \{\perp\}$ from H and $\{\vec{y}\}_{y \in \mathcal{Y}}$. For each column c_j of T_l for $j \in [B_l]$, compute $\widehat{ct}_j \leftarrow \mathsf{ThFHE.Enc}(pk, c_j)$ and send $\{\widehat{ct}_j\}_{j \in [Q_l]}$ for $Q_l := \lceil \frac{B_l}{k} \rceil$ to the leader server $\mathcal{S}_1$. After, $\mathcal{S}_1$ broadcasts $\{\widehat{ct}_j\}_{j \in [B_l]}$ to $\mathcal{S}_2, \ldots, \mathcal{S}_{l-1}$.

Computing Intersections:

- Upon receiving $\{\widehat{ct}_j\}_{j \in [B_l]}$, each $\mathcal{S}_i$ runs $\mathsf{Ext}(\widehat{ct}_j) \to \{\widehat{ct}_{j,1}, \ldots, \widehat{ct}_{j,k}\}$, and do the following:
  (1) For each $ct_{q,m} \in ct_{\mathcal{X}_i}$, compute $d_{j,q,m} \leftarrow \mathsf{ThFHE.Eval}(evk, \{ct_{q,m}, \widehat{ct}_{j,m}\}, -)$ for $j \in [B_l]$, where $-$ denotes the subtraction.
  (2) For $q \in [Q_i]$, compute $\widehat{d}_{j,q} \leftarrow \mathsf{ThFHE.Eval}(evk, \{d_{j,q,m}\}_{m=1}^k, C_\phi)$, $z_{j,q} \leftarrow \mathsf{ThFHE.Eval}(evk, \widehat{d}_{j,q}, C_{\mathsf{VAF}})$, $\hat{z}_{i,j} \leftarrow \mathsf{ThFHE.Eval}(evk, \{z_{j,q}\}_{q=1}^{Q_i}, C_\Sigma)$, and $\bar{z}_{i,j} \leftarrow \mathsf{ThFHE.Eval}(evk, \bar{z}_{i,j}, C_{\mathsf{RotAdd}})$.
  (3) For $j \in [B_l]$, sample $\vec{r}_{i,j} \xleftarrow{\$} \prod_{m=1}^k \mathbb{F}_p$ and encrypt $ct_{R,i,j} \leftarrow \mathsf{ThFHE.Enc}(pk, \vec{r}_{i,j})$. Finally, send$\{(\bar{z}_{i,j}, ct_{R,i,j})\}_{j \in [B_l]}$ to $\mathcal{S}_1$.

Aggregating the Results from Each Server:

- Upon receiving $\{(\bar{z}_{i,j}, ct_{R,i,j})\}_{j \in [B_l]}$, $\mathcal{S}_1$ aggregates the intersection results.
  (1) Compute $\bar{z}_j \leftarrow \mathsf{ThFHE.Eval}(evk, \{\bar{z}_{i,j}\}_{i=1}^{l-1}, C_\Sigma)$ and $\vec{ct}_{R,j} \leftarrow \mathsf{ThFHE.Eval}(evk, \{ct_{R,i,j}\}_{i=1}^{l-1}, C_\Sigma)$.
  (2) Compute $\bar{z}_j \leftarrow \mathsf{ThFHE.Eval}(evk, \{\bar{z}_j, \vec{ct}_{R,j}\}, \otimes)$, where $\otimes$ denotes homomorphic multiplication.
- After aggregation, $\mathcal{S}_1$ broadcasts $\{\bar{z}_j\}_{j \in [B_l]}$ to the client C and randomly selected $(\alpha - 1)$ servers, $\mathcal{S}_{n_1}, \ldots, \mathcal{S}_{n_{\alpha-1}}$ for $\{n_1, \ldots, n_{\alpha-1}\} \subset [l-1]$.

Partial Decryption and Interpreting the Output:

- For $i \in \{n_1, \ldots, n_{\alpha-1}, l\}$, compute $m_{i,j} \leftarrow \mathsf{ThFHE.PartialDec}(sk_i, \bar{z}_j)$. Then $\mathcal{S}_{n_1}, \ldots, \mathcal{S}_{n_{\alpha-1}}$ sends $\{m_{i,j}\}_{j=1}$ to C.
- C computes $m_j \leftarrow \mathsf{ThFHE.Combine}(pk, \{m_{n_1}, \ldots, m_{n_{\alpha-1}}, m_l\})$ for each $j \in [B_l]$. Initialize $\mathcal{I} = \emptyset$. For each $y \in \mathcal{Y}$, find $H(y) = \{i_{y,1}, \ldots, i_{y,k}\}$ and the column index j_y where y lie on. If m_{j_y}'s some of $i_{y,m}^{\text{th}}$ components is non-zero, then append y to $\mathcal{I}$. Return $\mathcal{I}$ as the intersection $\mathcal{Y} \cap (\bigcup_{i=1}^{l-1} \mathcal{X}_i)$.

Figure 5: Full protocol for solving DO-PSI.

Step 1: Setup and Server-Side Preprocessing. Before executing the protocol, we assume that the client C and servers $\mathcal{S}_1, \ldots, \mathcal{S}_{l-1}$ agree on parameters needed for the protocol, e.g., plaintext modulus p and ring dimension N for ThFHE, the size of each set element, the decryption threshold α, and the parameters to build a hashing table. Then, they jointly run ThFHE.Setup through an SMPC protocol or a trusted hardware to obtain the encryption key, evaluation key, and the secret key shares for each party. Each server sends the encryption key to the corresponding data owner. Each data owner encrypts its data and then sends the ciphertext to the corresponding server. More precisely, for each set element x, they parse it into k components $x_1, \ldots, x_k \in [0, p)$ such that $x = \sum_{i=1}^k x_i p^{i-1}$. By viewing the components as vectors $(x_1, \ldots, x_k)$, they run the vector-friendly hashing H in Section 5.3, obtaining a hashing table. For each sub-table that contains N/k bins, they pack consecutive k columns at once in the same ciphertext. Since there are k sub-tables for each k columns, they finally obtain k ciphertexts corresponding to each component of the parsed inputs. If the number of columns exceeds k, then they repeat the same procedure on the next columns.

Step 2: Query Processing. Next, on the client side, C first parses its inputs $y \in \mathcal{Y}$ into components $y_1, \ldots, y_k \in [0, p)$ such that $y = \sum_{i=1}^k y_i p^{i-1}$. Then, the client constructs a hashing table from these parsed set items. Finally, the client encrypts each column of the table and sends the ciphertexts to the leader server $\mathcal{S}_1$, who distributes them to other servers. For simplicity, we assume that the client's set is small enough to ensure that its hashing table has only one column with high probability, e.g., $1 - 2^{-40}$.

Step 3: Compute Intersections. Upon receiving a query ciphertext $\widehat{ct}$, each server computes the intersection by following the strategy in Section 5.3. More precisely, suppose that the server has ciphertexts $\{ct_i\}_{i \in [k]}$, where ct_i is an encryption of items belonging to the i^{th} sub-table. The server first extracts $\widehat{ct}_1, \ldots, \widehat{ct}_k$ with the query extraction method. The server then homomorphically subtracts $\widehat{ct}_i$ from ct_i and evaluates NPM on these subtractions. After evaluating the VAF on an input $\widetilde{ct}_i$, the resulting ciphertext contains 0 or 1. If the server holds more ciphertexts, the server sums all the VAF outputs to compress them into a single ciphertext. Next, the server exploits the rotation-and-addition technique to sum all values in slots. Additionally, the server also randomly samples a vector $\vec{r} \in \prod_{i=1}^N \mathbb{F}_p$ and encrypts it to obtain a ciphertext $ct_{R,i}$. The resulting ciphertexts are sent to the leader server $\mathcal{S}_1$. Since the remaining part of the protocol requires only one multiplication, each server can compress its ciphertexts using modulus switching to reduce the communication cost between servers.

Step 4: Aggregation, Partial Decryption, and Result Retrieval. After receiving ciphertexts from each server, the leader server first sums them, obtaining the two ciphertexts for the intersection result and the blinding factor, respectively. Then, the leader server multiplies them to hide the cardinality of matched items. To provide the decryption result to the client, the leader server designates $(\alpha - 1)$ servers and requests them to partially decrypt the aggregated ciphertext. The client also receives the aggregated ciphertext. All the partially decrypted ciphertexts are sent to the client, and the client finally runs ThFHE.Combine to recover the message. Finally, for each item, the client checks whether the corresponding positions in the hashing table with k duplicates are all zero or not, thereby determining the intersecting items over the entire set(s).

6 Theoretical Analysis

Efficiency Analysis. The major computational overhead of each server comes from the VAF evaluation. Since k columns can be evaluated at once, if we denote B_i as the maximum number of items across the whole bins in the $\mathcal{S}_i$'s hashing table, the total number of VAF evaluations is $\lceil \frac{B_i}{k} \rceil$. Since one VAF evaluation requires $2(k-1) + \log p$ multiplications, the total number of non-scalar multiplications is $\lceil \frac{B_i}{k} \rceil (2(k-1) + \log p)$. With probabilistic NPMs, we can reduce the number of multiplications by $2(w-1) + \log p$ for some $w < k$ when we allow a false positive probability of p^{-w}. After VAF evaluations, each server applies the rotation-and-addition technique, which requires $\lceil B_i/k \rceil + \log k$ additions and $\log k$ rotations. We can perform these operations on compressed ciphertexts via modulus switching for efficiency. Finally, the leader server aggregates the evaluated ciphertexts and blind factors independently and multiplies them. For the communication cost, per one column of the receiver's hashing table, only one ciphertext is communicated between any two parties. In addition, both the ciphertext from VAF evaluation and the constant random values for blind factors can be compressed via modulus switching. The final process for partial decryption and combining the results requires communicating α compressed ciphertexts. We note that the scale of B_i depends on the hashing algorithm. Because our vector-friendly hashing can exploit an existing hashing technique as a black-box, e.g., cuckoo hashing, we can directly follow prior results on the bound of B_i with respect to the server's set size from the PSI literature [16, 52].

How Many Servers Can Engage? To support the use cases described in Section 1, our protocol must be scalable to accommodate potentially thousands of servers. Moreover, since the evaluation of the VAF at each server can be performed in parallel, increasing the number of servers can improve the protocol's ability to handle larger aggregate datasets. We assume no duplicate items within an individual server's set, but these items may exist across the sets of different servers. Hence, the number of overlapped items for each client's item is at most the number of servers. Since the leader server sums the received ciphertexts over $\mathbb{F}_p$, to avoid overflow, at most $p-1$ servers can be engaged in our protocol.

Security Analysis. Several threshold FHE proposals support implementing SMPC under the semi-honest model [6, 23, 25, 54], whose corruption threshold solely depends on the decryption threshold. By following their proof strategy, we can prove the security of our protocol. Note that these proposals assume that the circuit evaluation process does not incur any communication between parties [6]; we can easily handle this issue because the colluding parties cannot decrypt ciphertexts communicated between servers. Hence, they are computationally indistinguishable from the random string, and all communications in our protocol are simulatable if the inputs and outputs of the protocol are provided. We prove the following statement, whose proof is provided in Appendix A.

Theorem 3. *The protocol in Fig. 5 securely implements $\mathcal{F}_{\text{DOPSI}}$ under the semi-honest model with at most $(\alpha - 1)$ colluding parties.*

Remark: Security Against Malicious Adversaries. Achieving malicious security can make our protocol more robust in practice. As discussed in FHE-based PSI works [15, 21], by applying an OPRF for each party's items before the protocol, one can achieve security against malicious clients while providing privacy against malicious servers, which is a somewhat relaxed notion of security compared to the standard definition [36]. Since our protocol has the same format as theirs, i.e., the client encrypts the query and lets the server do all computations in an encrypted domain during the protocol, we can also take advantage of the OPRF technique. Nevertheless, we need to introduce further assumptions that each data owner agrees on the same OPRF key or an additional entity that takes charge of OPRF evaluation. Hence, we focus on the semi-host model only; we leave addressing malicious adversaries as future work.

7 Experimental Analysis

We implement DO-PSI using C++17 and OpenFHE v1.2.3 [4] with a 64-bit backend. For reproducibility, we provide an anonymized GitHub link: https://github.com/whitesoonguh/DOPSI. All experiments were conducted on a machine with an AMD EPYC 7313P CPU (16 physical cores and 32 threads; 2.50GHz clock speed) and 128 GB of memory. Since we assume that each server has strong computational capability, we use all the cores in all experiments.

Experiment Setup. We used the threshold BFV scheme implemented in the OpenFHE library, using the plaintext modulus $p = 2^{16} + 1$ and the ring dimension $N = 2^{15}$. The remaining parameters, e.g., the ciphertext modulus, are chosen to ensure the 128-bit classical security [5]. For the security of underlying threshold FHE, we enabled NOISE_FLOODING_MULTIPARTY preset provided by OpenFHE, which adds a 120-bit random noise during partial decryption. In our experiments, likewise to [11, 41], we assume a one-time trusted initializer to process ThFHE.Setup. Note that the choice of threshold FHE instantiation is orthogonal to our DO-PSI; one can employ SMPC-based setup [6, 54] at a cost of higher communication overhead. We set the number of key shares $\alpha = \lfloor \frac{l}{2} \rfloor$.

To simulate the scenario with several servers, we measured the time elapsed for the single server's local computations; each server's computation is independent of the others, so it can be parallelized. We also simulate the cost of the aggregation for the leader server by using randomly generated ciphertexts with compressed to leave 3 modulus chains, reserving room for the noise flooding at the partial decryption phase. We account for the communication latency under both LAN and WAN settings. In LAN, we assume 10 Gbps bandwidth and 0.2ms round-trip time (RTT) latency, while our WAN assumes 200 Mbps bandwidth with an 80ms RTT latency.

Baseline Protocols for Comparison. As discussed earlier, several building blocks for designing PSI protocols, as well as the protocols based on them, are not applicable *as is* in DO-PSI. Therefore, for comparison, we select and implement three previous FHE-based protocols that can be straightforwardly adapted to DO-PSI, including APSI (asymmetrical-PSI) [21], PEPSI [52], and the protocol by Koirala et al. [45]. APSI is widely regarded as a strong baseline among FHE-based PSI protocols [45, 52, 74, 83]. PEPSI supports an additive aggregation on the leader server as they use an equality circuit-based approach for carefully encoded set items. Since the original implementations of APSI and PEPSI were in SEAL [71], we reimplemented the semi-honest version of these protocols in OpenFHE for a fair comparison. We also used the same threshold BFV scheme with the same amount of noise flooding as ours.

Although the original APSI and PEPSI assume unencrypted sender's sets, they can directly support encrypted sets without changing the protocol structure by encrypting the sender's set and performing ciphertext-ciphertext operations, so we reported the results from this setting. We set the size of each item to 80-bits and followed the recommended parameters by the authors. This ensures the false positive probability is less than 2^{-40} for all settings. In contrast, as the protocol by Koirala et al. [45] only supports items up to 25 bits due to the VAF design, we used their maximum supported parameters. We also emphasize that Koirala et al.'s protocol did extend their protocol to the PSI setting (i.e., $|\mathcal{Y}| > 1$), though one can implement it in an inefficient way by repeatedly running their protocol for each item. Hence, we compare it with others in the PMT setting only ($|\mathcal{Y}| = 1$). For the other protocols, we use cuckoo hashing with 3 hash functions. We set the parameters, e.g., the maximum number of items per bin, when the failure probability of constructing a hash table is 2^{-40}.

7.1 Overhead Analysis

We first analyze the overall overhead of our protocol. We assume that each set item is a 128-bit string. We view each 128-bit string item as an 8-dimensional $\mathbb{F}_p$-vector by parsing it into 16-bit chunks. In this setting, our protocol requires at most 19 depths because of NPM and VAF evaluations, and each ciphertext has a size of 7.34 MB and 1.57 MB before and after compression, respectively.

We measure each server's computational latency for computing the intersection and scale the number of total items being processed for each server, including dummy values in the hash table, from 2^{15} to 2^{24}. We test three methods, namely, (i) the VAF without NPM (Section 5.1; from the idea of Kim et al. [44]), (ii) the exact NPM, and (iii) the probabilistic NPM (Section 5.2). For the latter, we reduce 8-NPM to 4-NPM, which ensures the false positive rate from the NPM evaluation becomes less than 2^{-64}. We also measure the leader server's overhead for receiving and aggregating ciphertexts from the servers. We scale the number of servers from 2^4 to 2^{13}. Note that in our protocol, the aggregation and communication costs do not depend on the number of items per server.

The results are shown in Fig. 6. Each server's computational cost linearly scales with the number of items. NPM significantly reduces the latency as it can process more items at once than the method without NPM. With 2^{20} items, the NPM-free method takes 95.13 s, while the exact and probabilistic NPMs require 22.14 s and 15.86 s, achieving 4.30× and 6.00× improvements, respectively. The slight non-linear behavior for smaller items (2^{15}-2^{19}) comes from the fact that the NPM-based method can process 2^{15} items at once, thus not fully utilizing the available threads in our environment. On the other hand, aggregation and communication overheads scale linearly with the number of servers. The aggregation time is relatively cheaper than each server's individual computation, e.g., 0.94s for 1024 servers and 7.12s for 8192 servers. In contrast, the communication cost can dominate when the number of servers is huge, e.g., 2^{11} or more servers. The leader server exchanges about 10.48 MB with each server, 7.34 MB for distributing the query and 3.14MB for retrieving ciphertexts of intersection results and random masking. For 1024 and 8192 servers, the total communication cost amounts to 11.99 GB and 95.95 GB, respectively. Nevertheless, we later show that this is in fact significantly lower than prior works.

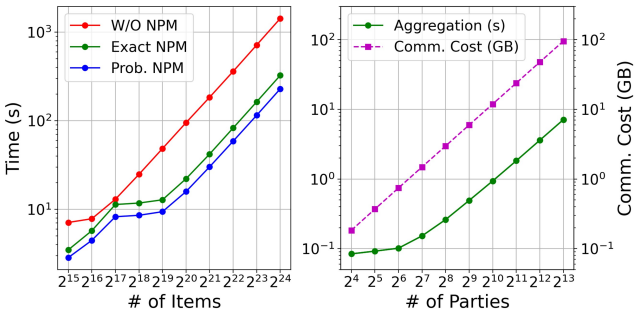


Figure 6: Local server's computation latency on computing intersection (left) and the aggregation/communication overheads of the leader server (right). Full numerical values are provided in Tab. 6 of Appendix E.

7.2 Comparison with State-of-the-art Protocols

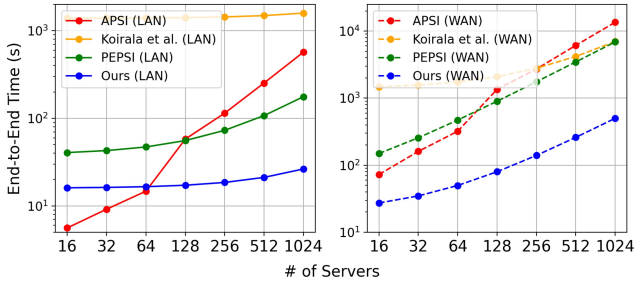
We now compare ours with the state-of-the-art FHE-based solutions, including APSI, PEPSI, and Koirala et al. [45]. For ours, we used the probabilistic NPM as it shows the best performance. We consider two settings: the PMT and PSI. As mentioned before, we compare Koirala et al.'s with others for the PMT setting only. For the PSI setting, we set the client's set size as 2048. To encode this, a hash table of 4096 bins from cuckoo hashing suffices. When the ring dimension exceeds 4096, we pack as many hash table columns as possible to fully utilize the plaintext slots. Conversely, if 4096 bins cannot be set, we use fewer bins and repeat the protocol as many times as needed to process all 2048 queries. We provide the detailed parameter settings of these protocols in the Appendix D.

We focus on the scalability with respect to the number of servers. Hence, we fix the number of items per server to 2^{20} and measure the end-to-end latency for processing queries. This includes the computation time for each server, the aggregation time, and the communication latency for various numbers of servers. For all protocols, we vary the number of servers from 16 to 1024.

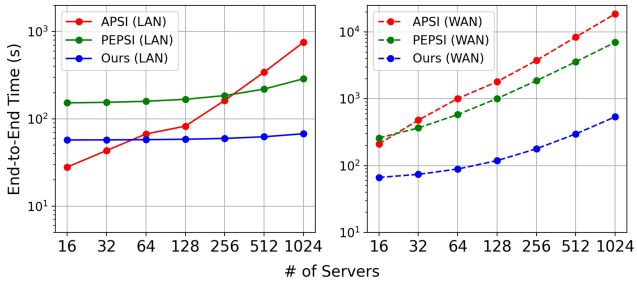
The results are provided in Fig. 7. Ours consistently outperforms the baselines, except for the LAN setting with a small number of servers, i.e., 16–64. Note that APSI is tailored for a two-party setting, so it performs better for a lower number of servers. However, APSI requires multiplications for aggregating results, which inflates the FHE parameters as the number of servers grows, resulting in larger

Table 2: The computation/communication costs of ours, APSI [21], and PEPSI [52] in the DO-PSI setting for varying number of servers l. Each server has an encrypted set of size 2^{20}, and the client's query set size is 2048. For each row, we emphasize the lowest and the second lowest latency or communication size as bold and underlined, respectively.

l	Total Comm. Size (GB)			Communication Latency (s)						Computation Latency (s)					
				LAN (10 Gbps)			WAN (200 Mbps)			Each Server's Computation			Leader Server's Aggregation		
	[21]	[52]	Ours	[21]	[52]	Ours	[21]	[52]	Ours	[21]	[52]	Ours	[21]	[52]	Ours
16	4.57	2.67	**0.18**	3.66	2.13	**0.14**	184.73	108.91	**9.34**	**21.22**	150.52	57.13	3.18	**0.08**	**0.08**
64	23.88	10.70	**0.74**	19.11	8.56	**0.59**	957.25	430.0	**31.72**	**25.8**	150.52	57.13	22.26	**0.11**	**0.11**
256	91.46	42.81	**2.98**	73.17	34.25	**2.38**	3660	1714	**121.25**	**17.04**	150.52	57.13	72.54	**0.26**	**0.26**
1024	455.81	171.3	**11.93**	364.65	137	**9.55**	18234	6852	**479.4**	**24.06**	150.52	57.13	366.03	**0.94**	**0.94**



(a) DO-PMT Setting; $|\mathcal{Y}| = 1$



(b) DO-PSI Setting; $|\mathcal{Y}| = 2048$

Figure 7: End-to-end latency of ours compared to FHE-based methods in LAN (10 Gbps) and WAN (200 Mbps) settings. Upper and lower figures correspond to DO-PMT and DO-PSI with $|\mathcal{Y}| = 2048$, respectively.

ciphertext sizes and unit costs for each operation. Additionally, APSI requires the client to send several powers of query ciphertexts to the servers. On the other hand, PEPSI and Koirala et al. support additive aggregations; hence, similar to ours, the computation cost for each server does not depend on the number of servers. This explains APSI's rapid performance degradation as the number of servers increases. Except for APSI, our protocol shows better latency than others across all the settings with different numbers of servers and network bandwidths. In particular, for the PMT setting, the protocol by Koirala et al. [45] shows higher overheads in all these settings, though they report that theirs outperforms APSI when the size of each server's set is limited to 2^{15}.

We also provide a detailed breakdown of latencies for the DO-PSI setting[2] in Table 2. Ours achieves the lowest communication and aggregation costs across various numbers of servers, while per-server computation cost remains comparable to APSI. Nevertheless, we can observe that ours outperforms the second-best one by a large margin, while the computation time for the single server stays comparable with APSI. We highlight that the proposed protocol takes a total latency of 67.62 s (537.43s, resp.) in LAN (WAN, resp.) settings for 1024 servers, which is 4.26–11.16 (13.03–34.65) × improvement in LAN (WAN, resp.) settings. We also provide the communication overhead per party in Table 3. Ours shows the lowest overhead for all types of communications. In particular, for the communication from the client to the server, ours shows a 17.30–49.86 × reduction compared to other protocols. This further highlights the scalability of our DO-PSI protocol with respect to the number of servers.

[2]Numerical values for the DO-PMT setting are provided in Tab. 7 of Appendix E.

Table 3: Communication cost per server (MB). Since the overhead for APSI depends on the total number of servers, we provide the results with 1024 servers. Note that Ours and PEPSI's incur the same overhead for both PMT and PSI.

Protocols	$C \to \mathcal{S}_i$	$\mathcal{S}_i \to \mathcal{S}_1$	$\mathcal{S}_1 \to C$	$\mathcal{S}_i \to C$
APSI [21] (PMT)	288.36	28.31	8.49	4.25
APSI [21] (PSI)	366.03	99.62	29.88	14.94
Koirala et al. [45]	127	4.2	4.2	2.1
PEPSI [52]	162.87	3.14	1.57	0.79
Ours	7.34	3.14	1.57	0.79

8 Conclusion

With the emergence of various real-world use cases requiring collaboration on sensitive data across different data sources, finding intersections among these sources in an encrypted form has become an important yet challenging problem. We address this problem by proposing a novel framework and several techniques to enhance efficiency. In particular, we proposed a novel tool called the nonzero-preserving mapping (NPM), which is of independent interest to improve the efficiency of other cryptographic protocols. Thanks to our techniques, our proposed protocol significantly outperforms existing solutions based on PSI both in computation and communication costs and shows superior scalability in terms of the number of servers, which can reach thousands or more.

Acknowledgments

This work was supported in part by the Culture, Sports and Tourism R&D Program through the Korea Creative Content Agency grant funded by the Ministry of Culture, Sports and Tourism in 2024 (RS-2024-00332210), the Institute for Information and Communications Technology Planning and Evaluation (IITP), grant funded by the Ministry of Science and ICT (MSIT), Republic of Korea (RS-2024-00399491), and National Science Foundation under CNS-2337321 and OAC-2312973. We are thankful to Mr. Kevin Vuong from the University of Notre Dame for helpful discussions.

References

[1] Aydin Abadi, Changyu Dong, Steven J Murdoch, and Sotirios Terzis. 2022. Multi-party updatable delegated private set intersection. In *Int. Conf. Financ. Cryptogr. Data Secur.* Springer, 100–119.

[2] Aydin Abadi, Sotirios Terzis, and Changyu Dong. 2015. O-PSI: delegated private set intersection on outsourced datasets. In *IFIP Int. Conf. ICT Syst. Secur. Priv. Prot.* Springer, 3–17.

[3] Aydin Abadi, Sotirios Terzis, Roberto Metere, and Changyu Dong. 2017. Efficient delegated private set intersection on outsourced private datasets. *IEEE TDSC* 16, 4 (2017), 608–624.

[4] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, et al. 2022. Openfhe: Open-source fully homomorphic encryption library. In *proceedings of the 10th workshop on encrypted computing & applied homomorphic cryptography*. 53–63.

[5] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, et al. 2021. Homomorphic encryption standard. *Protecting privacy through homomorphic encryption* (2021), 31–62.

[6] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. 2012. Multiparty computation with low communication, computation and interaction via threshold FHE. In *EUROCRYPT*. Springer, 483–501.

[7] American Hospital Association. 2025. Fast Facts on U.S. Hospitals 2025: 6,093 U.S. hospitals. https://www.aha.org/statistics/fast-facts-us-hospitals#:~:text=There%20are%206%2C093%20hospitals%20in,to%20provide%20visualizations%20for%20this Accessed: 2025-03-07.

[8] Saikrishna Badrinarayanan, Peihan Miao, Srinivasan Raghuraman, and Peter Rindal. 2021. Multi-party threshold private set intersection with sublinear communication. In *IACR PKC*. Springer, 349–379.

[9] Aslı Bay, Zekeriya Erkin, Jaap-Henk Hoepman, Simona Samardjiska, and Jelle Vos. 2021. Practical multi-party private set intersection protocols. *IEEE TIFS* 17 (2021), 1–15.

[10] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2023. Near-Optimal Oblivious Key-Value Stores for Efficient PSI,PSU and Volume-Hiding Multi-Maps. In *USENIX Security*. 301–318.

[11] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. 2018. Threshold cryptosystems from threshold fully homomorphic encryption. In *CRYPTO*. Springer, 565–596.

[12] Rob Bonta. 2022. California consumer privacy act (CCPA). *Retrieved from State of California Department of Justice: https://oag. ca. gov/privacy/ccpa* (2022).

[13] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *ACM TOCT* 6, 3 (2014), 1–36.

[14] Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. 2021. Efficient linear multiparty PSI and extensions to circuit/quorum PSI. In *ACM CCS*. 1182–1204.

[15] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from fully homomorphic encryption with malicious security. In *ACM CCS*. 1223–1237.

[16] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast private set intersection from homomorphic encryption. In *ACM CCS*. 1243–1255.

[17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT*. Springer, 409–437.

[18] Jung Hee Cheon, Wootae Kim, and Jai Hyun Park. 2022. Efficient homomorphic evaluation on large intervals. *IEEE TIFS* 17 (2022), 2553–2568.

[19] Eduardo Chielle, Homer Gamil, and Michail Maniatakos. 2021. Real-time private membership test using homomorphic encryption. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1282–1287.

[20] B Chor, O Goldreich, E Kushilevitz, and M Sudan. 1995. Private information retrieval. In *IEEE FOCS*. IEEE Computer Society, 41–41.

[21] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. 2021. Labeled PSI from homomorphic encryption with reduced computation and communication. In *ACM CCS*. 1135–1150.

[22] Consumer Financial Protection Bureau. 2024. Personal Financial Data Rights (12 CFR Part 1033). https://www.consumerfinance.gov/rules-policy/final-rules/personal-financial-data-rights/. Final Rule.

[23] Ronald Cramer, Ivan Damgård, and Jesper B Nielsen. 2001. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*. Springer, 280–300.

[24] Deloitte. 2024. Cloud Outsourcing Statistics: Trends and Insights. https://explodingtopics.com/blog/outsourcing-statistics Cited by Exploding Topics, includes global IT outsourcing statistics.

[25] S Dov Gordon, Feng-Hao Liu, and Elaine Shi. 2015. Constant-round MPC with fairness and guarantee of output delivery. In *CRYPTO*. Springer, 63–82.

[26] Thai Duong, Duong Hieu Phan, and Ni Trieu. 2020. Catalic: Delegated PSI cardinality with applications to contact tracing. In *ASIACRYPT*. Springer, 870–899.

[27] European Central Bank. 2024. Cloud Outsourcing in the Banking Sector: ECB Banking Supervision 2024 Analysis. https://www.bankingsupervision.europa.eu/press/publications/html/ssm.cloudoutsourcing2024.en.html Analysis of cloud outsourcing trends and expenditures in European banks.

[28] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012).

[29] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. 2005. Keyword search and oblivious pseudorandom functions. In *TCC*. Springer, 303–324.

[30] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. 2004. Efficient private matching and set intersection. In *EUROCRYPT*. Springer, 1–19.

[31] Ying Gao, Yuanchao Luo, Longxin Wang, Xiang Liu, Lin Qi, Wei Wang, and Mengmeng Zhou. 2024. Efficient Scalable Multi-Party Private Set Intersection (-Variants) from Bicentric Zero-Sharing. In *ACM CCS*. 4137–4151.

[32] Sanjam Garg, Mohammad Hajiabadi, Abhishek Jain, Zhengzhong Jin, Omkant Pandey, and Sina Shiehian. 2023. Credibility in Private Set Membership. In *IACR PKC*. Springer, 159–189.

[33] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2021. Oblivious key-value stores and amplification for private set intersection. In *CRYPTO*. Springer, 395–425.

[34] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *ACM STOC*. 169–178.

[35] Meng Hao, Weiran Liu, Liqiang Peng, Hongwei Li, Cong Zhang, Hanxiao Chen, and Tianwei Zhang. 2024. Unbalanced Circuit-PSI from Oblivious Key-Value Retrieval. In *USENIX Security*. 6435–6451.

[36] Carmit Hazay and Yehuda Lindell. 2008. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*. Springer, 155–175.

[37] HIPAA Journal. 2024. Healthcare Cloud Usage Grows But Protecting PHI Can Be a Challenge. https://www.hipaajournal.com/healthcare-cloud-usage-grows-but-protecting-phi-can-be-a-challenge/ Accessed: 2025-05-30.

[38] Ilia Iliashenko and Vincent Zucca. 2021. Faster homomorphic comparison operations for BGV and BFV. *PoPETs* 2021, 3 (2021), 246–264.

[39] Apple Inc. [n. d.]. Apple Password Monitoring. https://support.apple.com/guide/security/password-monitoring-sec78e79fc3b/web Accessed: 2025-08-22.

[40] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *ACM STOC*. 604–613.

[41] Aayush Jain, Peter MR Rasmussen, and Amit Sahai. 2017. Threshold fully homomorphic encryption. *Cryptology ePrint Archive* (2017).

[42] Guangshang Jiang, Hanlin Zhang, Jie Lin, Fanyu Kong, and Leyun Yu. 2024. Optimized verifiable delegated private set intersection on outsourced private datasets. *Computers & Security* 141 (2024), 103822.

[43] Florian Kerschbaum. 2012. Outsourced private set intersection using homomorphic encryption. In *ACM ASIACCS*. 85–86.

[44] Myungsun Kim, Hyung Tae Lee, San Ling, and Huaxiong Wang. 2016. On the efficiency of FHE-based private queries. *IEEE TDSC* 15, 2 (2016), 357–363.

[45] Nirajan Koirala, Jonathan Takeshita, Jeremy Stevens, and Taeho Jung. 2024. Summation-based Private Segmented Membership Test from Threshold-Fully Homomorphic Encryption. In *PoPETs*. Bristol, UK.

[46] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. 2017. Practical multi-party private set intersection from symmetric-key techniques. In *ACM CCS*. 1257–1272.

[47] Anunay Kulshrestha and Jonathan Mayer. 2021. Identifying harmful media in End-to-End encrypted communication: Efficient private membership computation. In *USENIX Security*. 893–910.

[48] Serge Lang. 2012. *Algebra*. Vol. 211. Springer Science & Business Media.

[49] Tong Liu, Zhen Huang, Jiaao Li, Jianyu Niu, Guoxing Chen, and Yinqian Zhang. 2024. SoK: Opportunities for Accelerating Multi-Party Computation via Trusted Hardware. In *2024 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 143–154.

[50] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On ideal lattices and learning with errors over rings. In *EUROCRYPT*. Springer, 1–23.

[51] Rasoul Akhavan Mahdavi and Florian Kerschbaum. 2022. Constant-weight PIR: Single-round keyword PIR via constant-weight equality operators. In *USENIX Security*. 1723–1740.

[52] Rasoul Akhavan Mahdavi, Nils Lukas, Faezeh Ebrahimianghazani, Thomas Humphries, Bailey Kacsmar, John Premkumar, Xinda Li, Simon Oya, Ehsan Amjadian, and Florian Kerschbaum. 2024. PEPSI: Practically Efficient Private Set Intersection in the Unbalanced Setting. In *USENIX Security*. 6453–6470.

[53] Carolyn Metnick. 2023. California Moves To Protect Medical Information Collected Through Reproductive And Sexual Health Applications. *Mondaq Business Briefing* (2023), NA–NA.

[54] Christian Mouchet, Juan Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. 2021. Multiparty homomorphic encryption from ring-learning-with-errors. *PoPETs* 2021, 4 (2021), 291–311.

[55] Muhammad Haris Mughees and Ling Ren. 2023. Vectorized batch private information retrieval. In *IEEE S&P*. IEEE, 437–452.
[56] Ofri Nevo, Ni Trieu, and Avishay Yanai. 2021. Simple, fast malicious multiparty private set intersection. In *ACM CCS*. 1151–1165.
[57] Rachel Nosowsky and Thomas J Giordano. 2006. The Health Insurance Portability and Accountability Act of 1996 (HIPAA) privacy rule: implications for clinical research. *Annu. Rev. Med.* 57, 1 (2006), 575–590.
[58] Office of the National Coordinator for Health Information Technology. 2024. Trusted Exchange Framework and Common Agreement (TEFCA). https://www.healthit.gov/topic/interoperability/policy/trusted-exchange-framework-and-common-agreement-tefca. Accessed: 2025-03-20.
[59] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. 2018. Varys: Protecting SGX enclaves from practical Side-Channel attacks. In *2018 Usenix Annual Technical Conference (USENIX ATC 18)*. 227–240.
[60] Rasmus Pagh and Flemming Friche Rodler. 2001. Cuckoo hashing. In *European Symposium on Algorithms*. Springer, 121–133.
[61] Seunghun Paik, Nirajan Koirala, Jack Nero, Hyunjung Son, Yunki Kim, Jae Hong Seo, and Taeho Jung. 2025. Scalable Private Set Intersection over Distributed Encrypted Data. *Cryptology ePrint Archive* (2025).
[62] Michael S Paterson and Larry J Stockmeyer. 1973. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.* 2, 1 (1973), 60–66.
[63] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security*. 515–530.
[64] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2018. Scalable private set intersection based on OT extension. *ACM TOPS* 21, 2 (2018), 1–35.
[65] Michael O Rabin. 2005. How to exchange secrets with oblivious transfer. *Cryptology ePrint Archive* (2005).
[66] Srinivasan Raghuraman and Peter Rindal. 2022. Blazing fast PSI from improved OKVS and subfield VOLE. In *ACM CCS*. 2505–2517.
[67] Sara Ramezanian, Tommi Meskanen, Masoud Naderpour, Ville Junnila, and Valtteri Niemi. 2020. Private membership test protocol with low communication complexity. *Digital Communications and Networks* 6, 3 (2020), 321–332.
[68] Forrester Research. 2023. State of Cloud in Healthcare, 2023. https://go.forrester.com/research/state-of-cloud-in-healthcare-202 Key findings on cloud adoption in healthcare.
[69] Peter Rindal and Phillipp Schoppmann. 2021. VOLE-PSI: fast OPRF and circuit-PSI from vector-OLE. In *EUROCRYPT*. Springer, 901–930.
[70] Gary Robinson, Sabine Dörry, and Ben Derudder. 2025. SWIFT: Trusted infrastructure for infrastructures. In *The Cambridge Global Handbook of Financial Infrastructure*. Cambridge University Press, 237–249.
[71] SEAL 2018. Microsoft SEAL (release 3.0). http://sealcrypto.org. Microsoft Research, Redmond, WA..
[72] What Is Intel SGX. [n. d.]. SgxPectre: Stealing Intel Secrets From SGX Enclaves via Speculative Execution. ([n. d.]).
[73] Nigel P Smart and Frederik Vercauteren. 2014. Fully homomorphic SIMD operations. *Des. Codes Cryptogr.* 71 (2014), 57–81.
[74] Yongha Son and Jinhyuck Jeong. 2023. PSI with computation or Circuit-PSI for Unbalanced Sets from Homomorphic Encryption. In *ACM ASIACCS*. 342–356.
[75] Statista. 2023. Number of FDIC-insured banks in the U.S. in 2023: 4,470. https://www.statista.com/statistics/184536/number-of-fdic-insured-us-commercial-bank-institutions/ Accessed: 2025-03-07.
[76] Yunqing Sun, Jonathan Katz, Mariana Raykova, Phillipp Schoppmann, and Xiao Wang. 2024. Actively Secure Private Set Intersection in the Client-Server Setting. In *ACM CCS*. 1478–1492.
[77] Sandeep Tamrakar, Jian Liu, Andrew Paverd, Jan-Erik Ekberg, Benny Pinkas, and N Asokan. 2017. The circle game: Scalable private membership test using trusted hardware. In *ACM ASIACCS*. 31–44.
[78] U.S. Congress. 1999. Gramm-Leach-Bliley Act (GLBA), Public Law 106-102. https://www.govinfo.gov/content/pkg/PLAW-106publ102/pdf/PLAW-106publ102.pdf. Enacted November 12, 1999.
[79] Adithya Vadapalli, Ryan Henry, and Ian Goldberg. 2023. Duoram: A Bandwidth-Efficient distributed ORAM for 2-and 3-party computation. In *USENIX Security*. 3907–3924.
[80] Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). *A practical guide, 1st ed., Cham: Springer International Publishing* 10, 3152676 (2017), 10–5555.
[81] Jelle Vos, Sikha Pentyala, Steven Golob, Ricardo Maia, Dean Kelley, Zekeriya Erkin, Martine De Cock, and Anderson Nascimento. 2024. Privacy-Preserving Membership Queries for Federated Anomaly Detection. *PoPETs* (2024).
[82] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. 2017. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *ACM CCS*. 2421–2434.
[83] Mingli Wu and Tsz Hon Yuen. 2023. Efficient unbalanced private set intersection cardinality and user-friendly privacy-preserving contact tracing. In *USENIX Security*. 283–300.
[84] Mingli Wu, Tsz Hon Yuen, and Kwan Yin Chan. 2024. O-Ring and K-Star: Efficient Multi-party Private Set Intersection. In *USENIX Security*. 6489–6506.
[85] Xinpeng Yang, Liang Cai, Yinghao Wang, Keting Yin, Lu Sun, and Jingwei Hu. 2024. Efficient Unbalanced Quorum PSI from Homomorphic Encryption. In *ACM ASIACCS*. 1003–1016.
[86] Xiaoyuan Yang, Xiaoshuang Luo, Xu An Wang, and Shuaiwei Zhang. 2018. Improved outsourced private set intersection protocol based on polynomial interpolation. *Concurrency and Computation: Practice and Experience* 30, 1 (2018), e4329.
[87] Yihao Yang, Yunbo Yang, Xiang Chen, Xiaolei Dong, Zhenfu Cao, and Jiachen Shen. 2024. DMPSI: Efficient Scalable Delegated Multiparty PSI and PSI-CA With Oblivious PRF. *IEEE Transactions on Services Computing* 17, 2 (2024), 497–508.

A Security Proof

We provide the security proof of the proposed protocol by following Definition 1. Due to the page limit, we provide the proof sketch only; the complete proof is available in the full version [61].

We first briefly introduce the (informal) security notions of threshold FHE. Semantic security means that for any adversary $\mathcal{A}$ who can access the partial secret keys of corrupted parties, the encryptions of any two messages are indistinguishable except with negligible probability. In addition, simulation security implies that given $(\alpha - 1)$ partial decryptions for distinct partial secret keys and the final decryption result m of ct, the remaining partial decryption of ct that gives the correct final decryption result can be efficiently simulated. We refer to [11] for formal definitions of them. Note that the ThFHE implementation by OpenFHE follows the construction by Asharov et al. [6], which satisfies all these security notions.

We now prove the following, which restates Theorem 3 in terms of Definition 1. We denote our protocol as π_{DOPSI} for simplicity.

Theorem 4. *Let* ThFHE *be a secure* (α, l)*-threshold FHE scheme used for instantiating* π_{DOPMT}*. Then* π_{DOPSI} *securely implements* $\mathcal{F}_{\text{DOPSI}}$ *under the semi-honest model with at most* $(\alpha - 1)$ *collusion.*

Proof Sketch. We need to show the existence of a simulator $\mathfrak{S}$ for the following cases of corruptions:

- **Case 1:** $\mathcal{S}_1, C \notin \mathcal{P}^*$,
- **Case 2:** $\mathcal{S}_1 \in \mathcal{P}^*$ and $C \notin \mathcal{P}^*$,
- **Case 3:** $\mathcal{S}_1 \notin \mathcal{P}^*$ and $C \in \mathcal{P}^*$
- **Case 4:** $\mathcal{S}_1, C \in \mathcal{P}^*$.

Note that we excluded the collusion by the data owner because it does not engage in the protocol after outsourcing.

Before providing simulators, we first show that the output of the real protocol is indistinguishable from that of the ideal functionality. In our protocol, probabilistic NPM and the blinding factor have failure probabilities p^{-w} and p^{-k}, respectively. We selected these parameters to ensure sufficiently low false positive ($\leq 2^{-64}$) and false negative probabilities ($\leq 2^{-128}$), thereby ensuring that the outputs of the real protocol and the ideal functionality are statistically indistinguishable with an advantage of at most 2^{-64}.

We now provide the core idea for constructing simulators. We assume that the ThFHE.Setup was successfully executed over all parties, resulting in a public key pk, an evaluation key evk, and a share sk_i of the secret key. We focus on simulating transcripts only, as all the internal randomness for each party during the protocol is independent of the input or internal state. The full constructions of simulators are provided in the full version [61].

Simulator for Case 1 & Case 2. In these cases, the corrupted parties' view contains the ciphertexts, including the query ciphertext

ct, the final one $\bar{z}$ for partial decryption, and the local intersection results $(\bar{z}_i, ct_{R,i})$ for Case 2 only, whose plaintexts are not available to these parties. Thus, due to the semantic security of ThFHE, $\mathfrak{S}$ can simulate them via random strings, which is indistinguishable from corresponding ciphertexts in the real execution of the protocol.

Simulator for Case 3 & Case 4. In these cases, now the corrupted parties view the intersection result and its partial decryptions. We will focus on simulating them only; the remaining views are ciphertexts that are already treated in Case 1 & Case 2.

First, $\mathfrak{S}$ can simulate the plaintext for final decryption m_{SIM} by filling a random element sampled from $\mathbb{F}_p$ on the positions chosen from vector-friendly hashing on $y \in \mathcal{Y} \cap (\cup_{i=1}^{l-1} \mathcal{X}_i)$, otherwise 0. Then, for $\bar{z}_{\text{SIM}} \leftarrow \mathsf{ThFHE.Enc}(pk, m_{\text{SIM}})$, $\mathfrak{S}$ can simulate the partial decryptions from servers as follows:

(1) $\mathfrak{S}$ first $(\alpha - 1)$ indices $I \subset \{1, \cdots, l-1\}$.
(2) Among selected servers, for corrupted ones, say S_i for some $i \in I$, compute $m_{i,\text{SIM}} = \mathsf{ThFHE.PartialDec}(sk_i, \bar{z}_{\text{SIM}})$.
(3) For non-corrupted $S_{j_1}, \ldots, S_{j_N}$, $j_1, \ldots, j_N \in I$, sample random values $m_{j_k,\text{SIM}}$ for $k \in [N-1]$, and compute $m_{j_N,\text{SIM}}$ such that $\mathsf{ThFHE.Combine}(pk, m_l, \{m_{i,\text{SIM}}\}_{i \in I}) = m_{\text{SIM}}$, where $m_l = \mathsf{ThFHE.PartialDec}(sk_l, \bar{z}_{\text{SIM}})$ for the secret key share sk_l of C.

Due to the simulation security of ThFHE, the simulated partial decryptions $m_{i,\text{SIM}}$ are indistinguishable from those in the protocol.

To sum up, $\mathfrak{S}$ defined as above can simulate the views of any corrupted parties of size up to $(\alpha - 1)$, completing the proof. □

B Optimality of Our Choice of 2-NPM

With a simple case analysis, we can show that our Pell equation-like choice is optimal among 2-NPMs in terms of efficiency.

Theorem 5. *Let $f : K \times K \to K$ be a 2-NPM over a field K. Then, evaluating it requires at least 2 multiplications and 1 depth.*

Proof. We first note that any function without depth consumption is a linear combination of indeterminates, which is of the form $ax + by$ for $a, b \in K$. One can easily find a non-trivial solution by setting $(x, y) = (-b, a)$. That is, at least 1 depth is required.

Now we consider functions using only 1 multiplication. Then the resulting function should be one of (1) $x^2 + ax + by$ or (2) $xy + ax + by$ for $a, b \in K$. For the former, we can observe that $(x, y) = (-a, 0)$ gives a nontrivial zero. On the other hand, for the latter case, we can observe that for $y \neq -a$ or 0, $(x, y) = \left(\frac{-by}{y+a}, y\right)$ gives a non-trivial solution. This completes the proof. □

Although this does not ensure that our NPMs from recursive composition are optimal, it shows that ours is a fairly good choice.

Constructing NPMs from Field Norm. For the choice of NPMs, one may attempt to seek a 2^l-NPM directly, rather than the recursive construction by Theorem 1. We provide a generic yet *inefficient* method using the field norm induced from the field extension.

Definition 4 (Field Norm). *Let K be a finite field and $L = K(\alpha)$ is a degree-d simple extension of K. For $\beta \in K$, we define the field norm $N_{L/K}(\beta)$ of β as the determinant of the K-linear operator $x \mapsto \beta x$. For variables $(x_1, \ldots, x_d)$, we define the field norm polynomial $f_{L/K} : K^d \to K$ by $f(x_1, \ldots, x_d) = N_{L/K}\left(\sum_{i=1}^{d} x_i \alpha^{i-1}\right)$.*

The properties of the field norm have been well studied, and our 2-NPM construction can be seen as the field norm polynomial obtained from the field norm operator $N_{K(\sqrt{\alpha})/K}(\cdot)$, where $\alpha \in K$ such that $X^2 - \alpha$ is irreducible over K. The following are well-known properties of the field norm, *e.g.*, see [48, Section V-§5].

- $N_{L/K}(0) = 0$ and for all $x \in K$, $N_{L/K}(x) \in F$.
- The restriction $N^*_{L/K} : K^* \to F^*$ becomes a group homomorphism between multiplicative groups.

These properties immediately imply that the field norm polynomial satisfies the definition of NPMs. However, despite its theoretical beauty, exploiting the field norm polynomial is rather intricate because we need to homomorphically compute the determinant of the given matrix. For example, we provide an example of 4-NPM over $K = \mathbb{F}_{65537}$ and $L = K(\sqrt[4]{3})$ obtained from this approach:

$$\begin{aligned} f(x_1, x_2, x_3, x_4) =& x_1^4 - 12x_1^2x_2x_4 - 6x_1^2x_3^2 \\ &+ 12x_1x_2^2x_3 + 36x_1x_3x_4^2 + 18x_2^2x_4^2 \\ &- 36x_2x_3^2x_4 + 3x_2^4 + 9x_3^4 - 27x_4^4. \end{aligned}$$

Note that this can be represented by a depth-2 arithmetic circuit:

$$\begin{aligned} f(x_1, x_2, x_3, x_4) =& 12A(x_2^2 + 3x_4^2) - 3(4Bx_1^2 + x_2^4 - 6B^2 + 9x_4^4) \\ &- 6x_3^2(6B + x_1^2) + x_1^4 + 9x_3^4, \end{aligned}$$

where $A = x_1x_3$ and $B = x_2x_4$. Nevertheless, computing this polynomial requires 14 multiplications, which is considerably more expensive than 6 multiplications from our approach. This is the reason why took an alternative approach for recursively composing the NPMs using the simple 2-NPM from Pell equation.

C Omitted Algorithms

In this section, we provide algorithms for NPMs in our protocol, which were omitted due to space constraints.

Efficient Evaluation of NPMs. Since we constructed the NPM in Section 5.2 through recursion, we can immediately obtain an algorithm to evaluate the NPM from the given input. In particular, we present a recursion-free algorithm based on the following observation: when traveling the computational graph for NPM evaluation, which is a binary tree, we can take a breadth-first search-like strategy. More precisely, we can think of the input vector $\vec{x} = (x_1, \ldots, x_k)$ as a leaf node of the computational graph and attempt to compute the nodes of the same depth first. For example, when evaluating 5-NPM with an input vector $\vec{x} = (x_1, x_2, x_3, x_4, x_5)$, the order of operations in our strategy becomes

(1) Compute $y_1 = f_2(x_1, x_2)$, $y_2 = f_2(x_3, x_4)$, and set $y_3 = x_5$.
(2) Compute $z_1 = f_2(y_1, y_2)$ and set $z_2 = y_3$.
(3) Return $f_2(z_1, z_2)$.

Here, the intermediate values of the same symbol are placed at the same level from the root node. In addition, if the number of variables at the current level is odd, then we just apply the identity mapping, i.e., 1-NPM, and ascend it to the next level. We describe the algorithm to evaluate the NPM using this strategy in Algorithm 1.

Query Extraction Method. To enable k parallel processing of columns in the hash table, we exploited a method to extract a query vector from the client. The algorithm is provided in Algorithm 2. Throughout describing algorithms, we use the following notations:

Algorithm 1 Recursion-Free Computation of NPM

Require: An input vector $\vec{x} = (x_1, \ldots, x_k) \in \prod_{i=1}^{k} K$ and a field element $\alpha \in K$ such that $X^2 - \alpha$ is irreducible over K.
Ensure: Evaluation of k-NPM $f_k(x_1, \cdots, x_k)$ in Theorem 2.
1: Initialize $n \leftarrow k$ and $z_i \leftarrow x_i$ for $i \in [k]$.
2: **while** $n > 1$ **do**
3: **for** $i = 1$ from $\lfloor \frac{n}{2} \rfloor$ **do**
4: Update $z_i \leftarrow z_{2i-1}^2 - \alpha \cdot z_{2i}^2$
5: **end for**
6: **if** n is odd **then**
7: $z_{\lceil \frac{n}{2} \rceil} = z_n$.
8: **end if**
9: Update $n \leftarrow \lceil \frac{n}{2} \rceil$.
10: **end while**
11: **return** z_1.

Algorithm 2 Query Extraction

Require: A ThFHE ciphertext $\widehat{ct}$ from a column of a vector-friendly hash table and ThFHE evaluation key evk. The number of compressed components k.
Ensure: Ciphertexts $\{ct_i\}_{i=1}^{k}$ where each ct_i is an encryption of a vector filled with the same some value, say y_i.
1: Set $m \leftarrow N/k$.
2: **for** i from 1 to k **do**
3: Set $\vec{m}_i \leftarrow \sum_{j=1}^{m} \vec{e}_{(i-1)\cdot m+j} \in \prod_{i=1}^{N} \mathbb{F}_p$.
4: $ct_i \leftarrow \mathsf{ThFHE.Eval}(evk, \{\widehat{ct}, \vec{m}_i\}, \times)$.
5: **for** j from 1 to $\log k$ **do**
6: Compute $tmp \leftarrow \mathsf{ThFHE.Eval}(evk, ct_i, \ll_{m \cdot 2^{j-1}})$ and update $ct_i \leftarrow \mathsf{ThFHE.Eval}(evk, \{ct_i, tmp\}, +)$.
7: **end for**
8: **end for**
9: **return** $\{ct_i\}_{i=1}^{k}$.

$\vec{e}_n$ denotes the one-hot vector whose n'th component is 1. + and $\times$ denote the addition and multiplication operators, respectively. We denote $\ll_n$ as a left-hand side rotation across message slots of the given threshold HE ciphertext. When invoking $\times$ for the message vector, we apply the NTT encoding on the vector in advance.

D Parameter Descriptions for Other Protocols

We provide brief descriptions and parameter selections for each protocol we compared. The FHE parameters are provided in Table 4.

Maximum Items per Bin for Cuckoo Hashing. We calculated the maximum number of bins in the Cuckoo hashing from the result by Chen et al. [16]. When we denote the event E as the case when at least one bin has more items than B when inserting N set elements into a hash table of m bins, we utilize the following formula

$$\Pr[E] \leq m \sum_{i=B+1}^{N} \binom{N}{i} \left(\frac{1}{m}\right)^i \left(1 - \frac{1}{m}\right)^{N-i}. \quad (1)$$

Since we are using cuckoo hashing with 3 hash functions, we put $3N$ in place of N in the above formula. For $m = 2^{11}, 2^{12}$ and the given N, we find the largest B when the R.H.S. of Eq. (1) is less than 2^{-40}, as provided in Table 5.

On the other hand, from the client side, we borrow the results from Chen et al. [16] on the maximum size of supported query sets. We observe that $|Y| \approx \frac{m}{3}$ ensures 40-bit statistical security from their empirical data. Hence, we select a conservatively smaller bound, $|Y| = \frac{m}{2}$, although it is expected that our protocol can support a slightly larger query set without incurring efficiency loss.

Table 4: Parameters for Each Protocol. N: Ring dimension. t: Plaintext modulus for BFV. Δ: Scaling factor for CKKS. Q: Ciphertext modulus. $|ct|$: Size of a fresh ciphertext. The numbers in APSI denote the number of servers.

Protocol		Scheme	N	t (Δ)	Depth	Q	$\lvert ct \rvert$ (MB)
APSI	2^4	BFV	2^{14}	$2^{16}+1$	5	$\approx 2^{360}$	1.57
	2^5		2^{14}	$2^{16}+1$	6	$\approx 2^{420}$	1.84
	2^6		2^{15}	$2^{16}+1$	7	$\approx 2^{480}$	4.19
	2^7		2^{15}	$2^{16}+1$	8	$\approx 2^{540}$	4.72
	2^8		2^{15}	$2^{16}+1$	9	$\approx 2^{540}$	4.72
	2^9		2^{15}	$2^{16}+1$	10	$\approx 2^{600}$	5.24
	2^{10}		2^{15}	$2^{16}+1$	11	$\approx 2^{660}$	5.76
PEPSI		BFV	2^{14}	$2^{16}+1$	6	$\approx 2^{420}$	1.84
Koirala et al.		CKKS	2^{18}	$\approx 2^{45}$	62	$\approx 2^{2870}$	127

Table 5: Maximum items per bin when inserting $3N$ items into $m = 2^{11}$ (Upper) and $m = 2^{12}$ (Lower) bins. The failure probability is at most 2^{-40}.

N	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}
B	113	185	314	552	1,001	1,862	3,528	6,785	13,190
N	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}
B	74	114	186	315	554	1,004	1,865	3,533	6,792

APSI [21]. In APSI, the server homomorphically evaluates a polynomial $f(x) = \prod_{y \in \mathcal{X}} (x - y)$ using some (encrypted) powers of x from the receiver. They utilized the Paterson-Stockmeyer algorithm [62] to reduce the number of non-scalar multiplications during polynomial evaluation. In addition, they introduced a new technique called the global postage stamp to reduce the number of powers sent by the receiver. Finally, they used the hashing technique to reduce the degree of the polynomial, while leveraging the batched evaluation of such polynomials constructed from each bin. To handle longer items, they simply chunk the given item into smaller chunks to fit into the plaintext modulus on the FHE and place them in contiguous bins of the hash table.

The computation and communication costs of APSI depend on the number of pre-computed powers by the receiver. They provided the presets for various sizes of servers' and receivers' sets, even for the membership test. However, we should carefully use them because of the discrepancy in settings between theirs and ours. First, their protocol in the DO-PMT scenario requires multiplicative aggregation, which results in additional depth consumption. In addition, because of the noise flooding technique, there should be an additional budget for the ciphertext modulus. For these reasons, we need to select a larger ring dimension than that from their presets under the same security level. This affects the number of maximum items per bin in their hash table and the structure of their global postage-stamp technique.

Table 6: The time taken by each server during the VAF evaluation for various scales of sets $|\mathcal{X}_i|$ for various types of VAFs. The aggregation time of the leader server and the total communication cost for various numbers of servers l. W/O, ENPM, and PNPM correspond to VAFs without NPM, with an exact NPM, and with a probabilistic NPM, respectively.

$\lvert\mathcal{X}_i\rvert$	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
W/O	7.11	7.82	12.98	24.79	48.50	95.13	184.04	360.72	716.93	1428.07
ENPM	3.48	5.74	11.34	11.77	12.81	22.14	42.02	82.84	163.68	325.29
PNPM	2.84	4.46	8.25	8.56	9.40	15.86	30.17	58.78	115.35	229.18

l	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}
Agg.	0.084	0.092	0.10	0.15	0.26	0.49	0.94	1.82	3.60	7.12
Comm.	0.18	0.37	0.75	1.50	3.00	5.99	11.99	23.98	47.97	95.95
LAN	0.15	0.30	0.60	1.20	2.40	4.80	9.59	19.19	38.38	76.76
WAN	11.37	18.86	33.86	63.84	123.81	243.74	483.62	963.36	1922.84	3841.82

Table 7: The computation/communication costs of ours, APSI [21], Koirala et al. [45], and PEPSI [52] in the DO-PMT setting for varying number of servers l. Each server has an encrypted set of size 2^{20}. For each row, we emphasize the lowest and the second lowest latency or communication size as bold and underlined, respectively.

l	Total Comm. Size (GB)				Communication Latency (s)								Computation Latency (s)							
					LAN (10 Gbps)				WAN (200 Mbps)				Each Server's Computation				Aggregation by the Leader Server			
	[21]	[45]	[52]	Ours	[21]	[45]	[52]	Ours	[21]	[45]	[52]	Ours	[21]	[45]	[52]	Ours	[21]	[45]	[52]	Ours
16	1.61	2.14	2.67	**0.18**	1.29	1.71	2.14	**0.15**	68.33	89.54	110.91	**11.37**	**3.75**	1388.16	38.42	15.86	0.55	1.45	**0.08**	**0.08**
64	7.70	8.58	10.70	**0.75**	6.16	6.87	8.56	**0.60**	312.18	347.16	432.01	**33.86**	**4.60**	1388.16	38.42	15.86	3.96	5.78	**0.11**	**0.11**
256	66.33	34.34	42.81	**3.00**	53.07	27.48	34.25	**2.40**	2657.22	1377.62	1716.44	**123.81**	**11.35**	1388.16	38.42	15.86	50.10	23.13	**0.26**	**0.26**
1024	330.78	137.39	171.25	**11.99**	264.62	109.91	137.01	**9.60**	13235.02	5499.48	6854.13	**483.62**	16.40	1388.16	38.42	**15.86**	288.34	92.53	**0.94**	**0.94**

Nevertheless, we observed that in the parameter presets for the membership test of size 1M provided by the authors[3], they used neither the global postage-stamp technique nor the Paterson-Stockmeyer algorithm. For this reason, we directly follow their parameter settings across the experiments, i.e., `"max_items_per_bin": 55`, and set an appropriate depth as the number of the server grows. In addition, we also implemented the encrypted database case by encrypting the hash tables in a column-wise manner.

PEPSI [52]. In PEPSI, each set item is encoded to the codeword set called *constant-weight codewords*, which is defined by $C_{l,\alpha} = \{\vec{x} \in \{0,1\}^l : \mathcal{HW}(\vec{x}) = \alpha\}$ for parameters l, α indicating the length and Hamming weight of codewords, respectively. As shown by [51], there is an efficiently computable, injective mapping from an integer $x \le \binom{l}{\alpha}$ to a codeword in $C_{l,\alpha}$. They built an FHE-based PSI protocol from an efficient algorithm to homomorphically check the equality of two codewords.

Their computation and communication costs strongly rely on the choice of l, α. Precisely, their protocol requires l plaintext-ciphertext multiplications, α ciphertext-ciphertext multiplications, while consuming $\log \alpha$ depths. If we assume that the server's set was encrypted, then the above cost becomes $(l + \alpha)$ ciphertext-ciphertext multiplications and $(\log \alpha + 1)$ depths. In addition, the receiver sends l ciphertexts during the protocol. We also note that the supporting length of items depends on $\binom{l}{\alpha}$. For these reasons, selecting appropriate l and α is crucial for the efficiency of their protocol.

In our experiments, to ensure the false positive probability $< 2^{-40}$, we select (l, α) such that $\binom{l}{\alpha} \ge 2^{80}$, while choosing α as the power of two. We empirically found that $(l, \alpha) = (89, 32)$ shows the best efficiency while satisfying $\binom{89}{32} \approx 2^{80.35}$. Hence, we used this parameter during experiments. We note that they employ permutation-based hashing [63] to reduce the length of each set item, which is crucial in enhancing the efficiency of the PEPSI. However, we disabled it here because we are interested in the membership test, and employing a hash table in PEPSI increases the number of operations in this case.

Koirala et al. [45]. In their protocol, the major computational overhead comes from evaluating their *approximated* VAF over a huge domain. To mitigate this, they first evaluate the domain extension polynomial (DEP) [18], which facilitates homomorphically evaluating the approximated polynomial in a large interval. More precisely, the DEP extends the domain $[-R, R]$ to $[-L^n R, L^n R]$ through recursively applying the domain extension procedure n times. They combined DEP with the Chebyshev polynomial approximation of the function $f_{K,S}(X) = K \cdot (1 - \tanh^2(S \cdot x))$ of degree c, where K and S are parameters to tune the shape of the function. After approximation, they do additional squaring and multiplying some constant ρ to separate the evaluation results of matched and non-matched cases, namely, $f_{K,S}(x) \mapsto \left(\rho \cdot (f_{K,S}(x))^{2^j}\right)^{2^k}$.

We used the parameters provided in their original paper, which were chosen through extensive empirical analysis. Since their protocol cannot support long items to ensure the desired level of false positive rate, we used the parameter that supports the largest domain, which is 25-bits. The precise parameter settings are as follows:

- DEP parameters: $L = 2.59$, $R = 6400$, $n = 9$.
- Chebyshev Approximation: $c = 247$, $K = 1.5$, $S = 10$.
- Squaring: $j = 12$, $k = 3$, $\rho = 2.7$.

In this setting, evaluating the VAF takes 62 depths. We used their official implementation result in GitHub: https://github.com/tjungND/caoe-cerberus-query.

E Full Numerical Experiment Data

We provide the full numerical data for Fig. 6 and Fig. 7 for the DO-PMT scenario in Section 7. The corresponding tables for each figure are presented in Tab. 6 and 7, respectively.

[3] https://github.com/microsoft/APSI/blob/b967a126b4e1c682b039afc2d76a98ea2c993230/parameters/1M-1.json