Low-Communication Updatable PSI from Asymmetric PSI and PSU

Guowei Ling, Peng Tang, Weidong Qiu

School of Cyber Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
{gw_ling,tangpeng,qiuwd}@sjtu.edu.cn.

Abstract. Private Set Intersection (PSI) allows two mutually untrusted parties to compute the intersection of their private sets without revealing additional information. In general, PSI operates in a static setting, where the computation is performed only once on the input sets of both parties. Badrinarayanan et al. (*PoPETs* 2022) initiated the study of Updatable PSI (UPSI), which extends this capability to dynamically updating sets, enabling both parties to securely compute the intersection as their sets are modified while incurring significantly less overhead than re-executing a conventional PSI. However, existing UPSI protocols either do not support arbitrary deletion of elements or incur high computational and communication overhead. In this work, we combine asymmetric PSI with Private Set Union (PSU) to present a novel UPSI protocol. Our UPSI protocol supports arbitrary additions and deletions of elements, offering a flexible approach to update sets. Furthermore, our protocol enjoys extremely low communication overhead, scaling linearly with the size of the update set while remaining independent of the total set size. We implement our protocol and compare it against state-of-the-art conventional PSI and UPSI protocols. Experimental results demonstrate that our UPSI protocol incurs $587 \sim 755\times$ less communication overhead than the recently proposed UPSI protocol (*AsiaCrypt* 2024) that supports arbitrary additions and deletions. Moreover, our UPSI protocol also has a significant advantage in running time, achieving an improvement of two to three orders of magnitude, especially in low-bandwidth environments due to the exceptionally low communication overhead. Specifically, with an input size of 2^{22} and the size of the addition/deletion set being 2^8, the existing UPSI protocol requires approximately 1650.45, 1789.5, and 3458.1 seconds at bandwidths of 200 Mbps, 50 Mbps, and 5 Mbps, respectively, whereas our UPSI protocol only requires around 3.16, 3.35, and 5.65 seconds under the same conditions. Our open-source implementation is available at: https://github.com/ShallMate/upsi.

1 Introduction

Private Set Intersection (PSI) enables two parties, each holding a private set, to compute the intersection of their sets while revealing nothing beyond the intersection itself (except for the size of their inputs). It has found broad application in a variety of scenarios, such as data mining on private data [1], measuring ad conversion rates [2], and private contact discovery [3]. Over the past decade, PSI has made remarkable progress, with numerous efficient PSI protocols having been proposed [4,5,6,7,8,9,10,11,12]. The most efficient PSI protocol [10] can compute the intersection in approximately one second for input sizes on the order of one million, requiring only tens of megabytes of communication overhead.

Despite the significant performance breakthroughs achieved by these efficient PSI protocols, they are restricted to a *static* setting. Specifically, if the set of either party updates, even by a single element, a complete PSI execution is required to obtain the updated intersection. However, in practical applications, the private sets of the PSI protocol are often subject to continuous updates, and the set intersection is computed multiple times as the sets grow or shrink over time. For example, a typical application of a PSI protocol is sample alignment prior to vertical federated learning [13], and the training data on both parties may need to be updated continuously or periodically. If each update to the sets requires re-executing the PSI protocol, it would result in a significant waste of resources.

A recent work by Badrinarayanan et al. [14] initiates the study of Updatable PSI (UPSI), which enables two parties to securely compute the intersection of updated sets without re-executing a conventional PSI. However, this UPSI protocol [14] only supports the addition of elements, while deletions are implemented through a periodic refresh mechanism, referred to as "weak deletion." Very recently, Badrinarayanan et al. [15] proposed a revised version of this protocol, which supports the addition and deletion of elements in UPSI. However, from the experimental results presented in Table 4 of [15], we observe that this protocol introduces new challenges, namely expensive computational and communication overhead. Specifically, it only outperforms a re-execution of the most efficient PSI protocol [10] under particular conditions, namely when the total input is very large (i.e., 2^{22}), the update set is minimal (i.e., 2^4), and the available bandwidth is significantly constrained (i.e., 5 Mbps). This means that, unless under the aforementioned particular conditions, executing this UPSI protocol [15] that supports deletion will take more time than re-executing the most optimal conventional PSI [10]. Therefore, this raises a natural question:

Could we construct a UPSI protocol that supports arbitrary additions and deletions of elements while ensuring faster performance than re-executing a conventional PSI in most cases, rather than being limited to highly specific parameters/bandwidths?

1.1 Our Results

This work addresses the above question by constructing a UPSI protocol that allows both parties to arbitrarily add or delete elements from their input sets while incurring significantly lower computational and communication overhead than the deletion-supporting UPSI protocol in [15], making the execution of our UPSI protocol faster than re-executing the state-of-the-art conventional PSI [10] in most cases. Our UPSI is inspired by the UPSI framework based on structured encryption [16,17] proposed by Agarwal et al. [18]. However, our UPSI protocol does not rely on structured encryption but instead borrows the idea of asymmetric PSI (i.e., unbalanced PSI) and requires any efficient Private Set Union (PSU) protocol. Additionally, the UPSI protocol proposed by Agarwal et al. [18] provides only a complexity analysis, with neither experimental results nor code implementation available, leaving its concrete performance unclear. In Table 1, we compare our UPSI protocol with previous UPSI protocols [14,15,18] in terms of support for addition and deletion of elements, computational and communication complexity, availability of code implementations and experimental results, and practical communication.

Table 1: Summary of our results in comparison to previous work, including support for addition and deletion of elements, computational and communication complexity, whether code implementations and experiments, and practical communication ratio are provided. N denotes the size of the entire sets, and N_u denotes the size of the updated sets. t denotes the number of updates when parties refresh their sets with weak deletion. k denotes a constant. Since BMX22 [14] and ACG$^+$24 [18] did not provide code, we cannot compare practical communication with them.

Protocol	Addition/Deletion	Comp. Complexity	Comm. Complexity	Code/Experiment	Practical Communication Ratio
BMX22 (addition-only) [14]	Addition	$\mathcal{O}(N_u)$	$\mathcal{O}(N_u)$	Experiment	-
BMX22 [14]	Weak Deletion	$\mathcal{O}(N_u \cdot t)$	$\mathcal{O}(N_u \cdot t)$	Experiment	-
BMS$^+$24 (addition-only) [15]	Addition	$\mathcal{O}(N_u \cdot \log N)$	$\mathcal{O}(N_u \cdot \log N)$	Code & Experiment	5 - 6
BMS$^+$24 [15]	Addition & Deletion	$\mathcal{O}(N_u \cdot \log^2 N)$	$\mathcal{O}(N_u \cdot \log^2 N)$	Code & Experiment	587 - 755
ACG$^+$24 [18]	Addition & Deletion	$\mathcal{O}((\log N)^k)$	$\mathcal{O}(N_u)$	-	-
Ours	Addition & Deletion	$\mathcal{O}(N_u \cdot \log N)$	$\mathcal{O}(N_u)$	Code & Experiment	1

To the best of our knowledge, our work is the first UPSI protocol to provide comprehensive experimental results, an open-source implementation, and support for adding and deleting elements while guaranteeing efficient performance. Furthermore, we present our main results in terms of experiments, computation, communication, and end-to-end.

• **Experiments**. We implement our UPSI protocol and provide a comprehensive report on its performance under various input sizes, update set sizes, and bandwidth conditions to strengthen reader confidence in our work. Moreover, we also compare our protocol with the state-of-the-art UPSI and conventional PSI protocols, demonstrating the performance advantages of our UPSI protocol. Finally, we evaluate the update size threshold at which re-executing the conventional PSI becomes more efficient than our UPSI protocol, which is an important experiment that has been overlooked in previous works [14,15,18].

• **Computation**. Our UPSI protocol outperforms existing UPSI protocols in terms of computational overhead. It reduces computation overhead by $194 \sim 2015\times$ compared to the state-of-the-art UPSI protocol [15] that supports both addition and deletion and most $32\times$ faster compared to the version that supports only addition.

• **Communication**. Our protocol reduces communication overhead by a factor of $587 \sim 755$ compared to the state-of-the-art UPSI protocol [15] that supports both addition and deletion. Additionally, it achieves a $5 \sim 6\times$ reduction compared to the version that supports only addition. Additionally, our protocol has a communication overhead that is $10 \sim 2066\times$ less than that of the state-of-the-art conventional PSI protocol [10] in all settings.

• **End-to-End**. Our UPSI demonstrates an advantage in the end-to-end running time with all bandwidth settings. For example, our UPSI protocol can be up to $1684\times$ faster than the state-of-the-art UPSI protocol [15] that supports both addition and deletion with a bandwidth of 200 Mbps. Furthermore, our protocol can be $8 \sim 176\times$ faster than the state-of-the-art conventional PSI protocol [10] with a bandwidth of 5 Mbps.

1.2 Technical Overview

From a high-level perspective, our UPSI protocol requires executing an asymmetric PSI protocol twice, with computational and communication overhead depending solely on the party with the smaller set. Unlike previous mainstream asymmetric PSI protocols [19,20,21,22], our protocol outputs the result to the party with the larger set. Furthermore, this process does not require fully homomorphic encryption [23,24] to protect data privacy and hashing technologies [25] for element alignment between both parties. Meanwhile, our protocol requires a PSU protocol as a foundational component, for which several efficient solutions exist, such as those presented in [26,27,28,29,30]. Now, let us introduce the core content of our UPSI protocol step by step from a high-level perspective. We would like to emphasize that certain specific computational steps are omitted here (but are required in the actual protocol) to maintain clarity for the reader.

Initialization. Let $\mathcal{P}_X$ and $\mathcal{P}_Y$ denote the parties holding the original sets X and Y, respectively. $\mathcal{P}_X$ and $\mathcal{P}_Y$ can execute an existing two-party PSI protocol as a base protocol, such as those in [6,8,10], to obtain the intersection $I = X \cap Y$.

Deletion. Let $\mathcal{P}_X$ and $\mathcal{P}_Y$ intend to delete the sets X^- and Y^-, respectively.

Addition. Let $\mathcal{P}_X$ and $\mathcal{P}_Y$ intend to add the sets X^+ and Y^+, respectively.

Compute the intermediate intersection. Let the updated sets of $\mathcal{P}_X$ and $\mathcal{P}_Y$ be denoted as X_1 and Y_1, respectively, where $X_1 = (X \setminus X^-) \cup X^+$ and $Y_1 = (Y \setminus Y^-) \cup Y^+$. $\mathcal{P}_X$ and $\mathcal{P}_Y$ execute our asymmetric PSI protocol using X_1 and Y^+, with $\mathcal{P}_X$ obtaining $T = Y^+ \cap X_1$. Similarly, $\mathcal{P}_X$ and $\mathcal{P}_Y$ execute the asymmetric PSI protocol again using X^+ and Y_1, with $\mathcal{P}_Y$ obtaining $V = X^+ \cap Y_1$. Note that the computational and communication overhead of this step is linear in $|X^+|$ and $|Y^+|$, and independent of the size of the entire sets held by both parties.

Compute the intermediate union. $\mathcal{P}_X$ and $\mathcal{P}_Y$ use T and V as inputs, respectively, and invoke an existing PSU protocol, such as those in [26,27,28,29,30], with both parties obtaining $U = T \cup V$. Subsequently, $\mathcal{P}_X$ and $\mathcal{P}_Y$ locally compute $X^- \cap I$ and $Y^- \cap I$, respectively. Both parties invoke the PSU protocol again, and each receives $U' = (X^- \cap I) \cup (Y^- \cap I)$.

Compute the updated intersection. Finally, each party can locally compute $I_1 = (I \backslash U') \cup U$ as the result of $X_1 \cap Y_1$.

It can be observed that our UPSI protocol requires executing a conventional PSI protocol as a base PSI during the initial intersection between the two parties. After both parties update their sets, the UPSI protocol can be executed repeatedly by following the same steps (except for initialization). Our protocol is formally described in Figure 4 and is proven secure in the semi-honest model. It achieves worst-case communication complexity that grows linearly with the size of the update sets and computation complexity that grows poly-logarithmically with the size of the entire sets and linearly with the size of the update sets.

2 Related Work

We provide a brief review of PSI, asymmetric PSI, PSU, and UPSI, with the first three serving as foundational components needed for this paper, while UPSI is the goal we aim to achieve.

PSI. Early PSI protocols were primarily constructed based on the DH key agreement [31,32]. Due to continuous optimizations in scalar multiplication on elliptic curves, primarily through two efficient curves, Curve25519 [33] and Four$\mathbb{Q}$ [34], the efficiency of DH-based PSI protocols has been significantly improved. The advantages of DH-based PSI protocols are their ease of implementation and relatively low communication overhead. Consequently, some modern DH-based PSI protocols [35,36,37] have been proposed in recent years. The state-of-the-art DH-based PSI protocol, proposed by Rosulek et al. [35], is known to be one of the fastest and most communication-efficient protocols for small input sizes. Pinkas et al. [4] constructed an efficient PSI protocol based on Oblivious Transfer (OT) extension [38], which can be considered the origin of highly efficient OT-based PSI protocols. Since then, many efficient PSI protocols [5,6,7,8] have been developed, with KKRT16 [6] and CM20 [8] being among the most efficient. However, compared to DH-based PSI protocols, these protocols sacrifice communication efficiency in exchange for higher computational efficiency. Fortunately, this changed with the advent of OKVS [7,39], which provides a convenient way to represent private sets and facilitates subsequent intersection calculations. Initially, OKVS was introduced to address the challenge of achieving maliciously secure PSI protocols, as cuckoo hashing [40] was unsuitable for this purpose. For further details, please refer to [39]. Consequently, the communication overhead of the first OKVS-based PSI protocols was high. Garimella et al. [41] addressed this issue, and Rindal et al. [9] subsequently combined OKVS with a VOLE protocol [42] to create an efficient PSI protocol with very low communication. Shortly afterward, Raghuraman et al. [10] improved the OKVS in [9] and combined it with a more efficient VOLE protocol [43], resulting in a state-of-the-art PSI protocol with extremely low computational and communication overhead.

Asymmetric PSI. Asymmetric PSI, also known as unbalanced PSI, is a special case of PSI where the set held by one party is significantly smaller than the set held by the other. In general, the intersection in an asymmetric PSI protocol is obtained by the party with the smaller input set. The current asymmetric PSI protocols are primarily constructed based on fully homomorphic encryption [23,24]. Chen et al. [19] introduced optimizations to reduce the multiplicative depth of the function evaluated homomorphically, thereby enhancing efficiency. Furthermore, Chen et al. [20] and Cong et al. [21] employ a combination of OPRF and fully homomorphic encryption, building upon [19]. This not only enhances performance but also extends security to the malicious model and enables the protocol to handle elements of arbitrary bit length. Recently, Mahdavi et al. [22] further optimized unbalanced PSI through a combination of constant-weight encoding and hashing techniques.

PSU. Currently, known PSU protocols are generally constructed using two approaches: additively homomorphic encryption [44] and OT extension [38,43]. We primarily focus on OT-based PSU protocols due to their higher efficiency. Kolesnikov et al. [26] proposed the first efficient PSU protocol, which features good practical performance and is several orders of magnitude faster than previous PSU protocols. Subsequently, Garimella et al. [27] proposed a new PSU protocol based on oblivious switching [45]. Jia et al. [28] also proposed two shuffle-based PSU protocols built on the oblivious switching, which they referred to as the Permute + Share subprotocol. Consequently, the performance of their protocols is similar to that of [27]. Recently, Zhang et al. [29] proposed two general constructions for PSU protocols with linear computational and communication complexity.

UPSI. In contrast to the primitives mentioned above, research on UPSI has only begun in the past two years, initially proposed and defined by Badrinarayanan et al. [14]. The UPSI protocol in [14] essentially only supports the addition of elements, while deletion is achieved through a periodic refreshing method, which they refer to as weak deletion. Additionally, Abadi et al. [46] proposed a delegated UPSI protocol, but this requires the involvement of a third-party cloud server. Recently, Badrinarayanan et al. [15] appear to have addressed this issue by proposing new UPSI protocols that support both the addition and deletion of elements. However, we observe that their protocols only demonstrate an advantage under very restrictive conditions, namely, very low bandwidth, tiny update set sizes, and large input sets. Around the same time, Agarwal et al. [18] also propose a UPSI protocol based on structured encryption [16,17]. However, they do not provide experimental results and code, only theoretical analysis. As a result, the concrete performance of this protocol remains unclear.

3 Preliminaries

3.1 Notation

Let p be a large prime. Let $\mathbb{G}$ be a group of prime order p with generator g. Let $\mathsf{H} : \{0,1\}^* \to \mathbb{G}$ be a hash function. Let $\mathcal{P}_X$ and $\mathcal{P}_Y$ denote the parties holding the original sets X and Y, respectively. Each party intends to add sets X^+ and Y^+, and delete sets X^- and Y^-. The updated sets are denoted as

$X_1 = (X \setminus X^-) \cup X^+$ and $Y_1 = (Y \setminus Y^-) \cup Y^+$. Let the input size and update size for both parties be N and N_u, respectively.

3.2 Private Set Intersection

The ideal functionality of the PSI protocol is presented in Figure 1. In this protocol, both parties, $\mathcal{P}_X$ and $\mathcal{P}_Y$, hold private sets, denoted as X and Y, respectively. They aim to compute the intersection $X \cap Y$ without revealing any additional information. Specifically, each party should only learn $|X|$, $|Y|$, and $X \cap Y$, while remaining unaware of $Y \backslash X$ and $X \backslash Y$. The concept of PSI originated with Meadows et al. [31], initially inspired by the DH key agreement. In recent years, efficient PSI protocols [6,7,8,9,10] have primarily relied on the oblivious pseudorandom function protocol.

$\mathcal{F}_{\mathsf{PSI}}$

There are two parties, $\mathcal{P}_X$ and $\mathcal{P}_Y$, who hold private sets X and Y, respectively. Both parties receive the intersection $I = X \cap Y$ as the output.

Fig. 1: Ideal functionality $\mathcal{F}_{\mathsf{PSI}}$.

3.3 Private Set Union

We use the PSU protocol as one of the core components of our UPSI protocol. In contrast to the rapid development of PSI over the past decade, PSU has only recently started to attract attention. Fortunately, several highly efficient OT-based PSU protocols have already been proposed [26,27,28,29,30]. As shown in Figure 2, in the PSU protocol, $\mathcal{P}_X$ holds X and $\mathcal{P}_Y$ holds Y, and they securely compute $X \cup Y$ for both parties without revealing $X \cap Y$. Similar to the PSI protocol, the PSU protocol also reveals the input sizes of both parties, i.e., $|X|$ and $|Y|$.

$\mathcal{F}_{\mathsf{PSU}}$

There are two parties, $\mathcal{P}_X$ and $\mathcal{P}_Y$, who hold private sets X and Y, respectively. Both parties receive the union $U = X \cup Y$ as the output.

Fig. 2: Ideal functionality $\mathcal{F}_{\mathsf{PSU}}$.

3.4 Updatable Private Set Intersection

UPSI is a variant of PSI that allows both parties to compute the intersection on dynamically updating sets. The concept of UPSI was recently introduced by Badrinarayanan et al. [14], who also provided an improved version [15]. In this work, we define the UPSI protocol that supports both addition and deletion operations in Figure 3. In our definition, we require an ideal PSI to perform the initial intersection between both parties. In other words, in each updated intersection computation, our UPSI protocol essentially operates on the updated input sets and the intersection obtained from the most recent intersection computation.

$\mathcal{F}_{\mathsf{UPSI}}$

There are two parties, $\mathcal{P}_X$ and $\mathcal{P}_Y$, who hold private sets X and Y, respectively. Both parties have obtained $I = X \cap Y$ using an ideal $\mathcal{F}_{\mathsf{PSI}}$. $\mathcal{P}_X$ updates its set to $X_1 = (X \setminus X^-) \cup X^+$ by deleting X^- and adding X^+. Similarly, $\mathcal{P}_Y$ updates its set to $Y_1 = (Y \setminus Y^-) \cup Y^+$ by deleting Y^- and adding Y^+. Both parties receive the updated intersection $I_1 = X_1 \cap Y_1$ as the output. If both parties need to perform the next UPSI, they set $I = I_1$, $X = X_1$, $Y = Y_1$, and then use the new X^+, X^-, Y^+, and Y^- to obtain the new X_1 and Y_1 to compute the updated intersection $I_1 = X_1 \cap Y_1$.

Fig. 3: Ideal functionality $\mathcal{F}_{\mathsf{UPSI}}$.

3.5 Secure Model

The semi-honest model [47] is used in this paper, where a semi-honest adversary may corrupt parties before executing the protocol. In other words, the parties in the protocol will honestly execute the protocol as agreed. However, a party corrupted by the adversary will attempt to extract additional information from its view. Consider a two-party protocol for computing the function $\mathcal{F}_{\Pi}(X, Y)$, where $\mathcal{P}_X$ has private input X and $\mathcal{P}_Y$ has private input Y. For $\mathcal{P}_X$, let $\mathsf{VIEW}^{\Pi}_{\mathcal{P}_X}(1^\kappa, X, Y)$ denote the view of party $\mathcal{P}_X$ during an honest execution of Π on input X. This view consists of the input, the random tape, and all messages exchanged by $\mathcal{P}_X$ as part of the Π protocol. Similarly, $\mathsf{VIEW}^{\Pi}_{\mathcal{P}_Y}(1^\kappa, X, Y)$ represents the view of $\mathcal{P}_Y$. I_X and I_Y denote the outputs of $\mathcal{P}_X$ and $\mathcal{P}_Y$ in $\mathcal{F}_{\Pi}(X, Y)$, respectively.

Definition 1. *(Semi-Honest Model)* [47]. Π *securely realizes* $\mathcal{F}_{\Pi}$ *in the presence of semi-honest adversaries if there exists two simulators* $\mathsf{SIM}^{\Pi}_{\mathcal{P}_X}$ *and* $\mathsf{SIM}^{\Pi}_{\mathcal{P}_Y}$ *such that*

$$\mathsf{SIM}^{\Pi}_{\mathcal{P}_X}(1^\kappa, X, I_X) \approx \mathsf{VIEW}^{\Pi}_{\mathcal{P}_X}(1^\kappa, X, Y),$$
$$\mathsf{SIM}^{\Pi}_{\mathcal{P}_Y}(1^\kappa, Y, I_Y) \approx \mathsf{VIEW}^{\Pi}_{\mathcal{P}_Y}(1^\kappa, X, Y),$$

where $\approx$ denotes computational indistinguishability with respect to the security parameter κ.

4 Updatable PSI from Asymmetric PSI and PSU

In this section, we formally describe our updatable PSI protocol, which supports arbitrary additions and deletions based on our asymmetric PSI and any efficient PSU protocol. Our UPSI protocol can be completed in a constant number of rounds. We first describe the components required for our UPSI protocol, then provide the specific construction of the protocol and demonstrate its correctness. Following that, we conduct a comprehensive analysis of the complexity. Finally, we prove that our protocol is secure against semi-honest adversaries.

4.1 Component Description

We begin by introducing the individual components and then combine them to form our UPSI protocol.

Base PSI. We require an efficient conventional two-party PSI protocol as a base PSI to perform the initial intersection between the two parties. For efficiency in the overall protocol, we recommend using the state-of-the-art conventional PSI protocol [10] to accomplish this task. Note that the performance of the base PSI is independent of the performance of the subsequent updated intersection computations, and the base PSI only needs to be executed once. Therefore, in practice, other efficient PSI protocols can also be used as the base PSI here, such as those in [6,7,8,9], which will only result in a certain performance loss during the initial intersection computation and will not affect the performance of subsequent updated intersection computations.

Asymmetric PSI. Our UPSI requires such an asymmetric PSI: the party with the larger input set obtains the intersection, while the computational and communication overhead only depends on the smaller set. For example, when $|X| \gg |Y|$, $\mathcal{P}_X$ and $\mathcal{P}_Y$ execute the asymmetric PSI protocol, with $\mathcal{P}_X$ receiving $X \cap Y$ while $\mathcal{P}_Y$ learns nothing except for $|X|$.

We achieve such an asymmetric PSI protocol based on the DH agreement. Specifically, $\mathcal{P}_X$ and $\mathcal{P}_Y$ randomly sample k'_x and k'_y from $\mathbb{Z}_p^*$, respectively. $\mathcal{P}_X$ and $\mathcal{P}_Y$ can locally compute $H_X = \{\mathsf{H}(x)^{k_x}\}$ and $H_Y = \mathsf{H}(y)^{k_y}$, respectively, for all $x \in X$ and $y \in Y$. Then, $\mathcal{P}_Y$ can send H_Y to $\mathcal{P}_X$, allowing $\mathcal{P}_X$ to compute $E_Y = \{h_y^{k_x}\}$ for all $h_y \in H_Y$ and send it back to $\mathcal{P}_Y$. Subsequently, $\mathcal{P}_Y$ locally computes $H'_Y = \{e_y^{k_y^{-1}}\}$ for all $e_y \in E_Y$ and sends it to $\mathcal{P}_X$. Finally, $\mathcal{P}_X$ obtains the intersection $X \cap Y$ by computing $H_X \cap H'_Y$. It is evident that the communication overhead of the above process is solely linear with respect to the size of Y. Moreover, since H_X can be precomputed independently by $\mathcal{P}_X$, the computational overhead in the above process is also linearly dependent on the input size of the party with the smaller set.

We would like to emphasize that if an alternative asymmetric PSI protocol is available, where the party with the larger set receives the intersection, and both computational and communication overheads scale linearly with the size of the smaller set, it can directly replace the asymmetric PSI described here. Additionally, we also note that the asymmetric PSI protocol proposed by Angelou et al. [48] is similar to the one presented here. However, in [48], the party with the smaller set receives the intersection, which does not meet the requirements for constructing UPSI in our context. Furthermore, due to the integration of the Bloom filter in [48], the protocol suffers from false positives and makes updating the set (either adding or deleting elements) challenging.

PSU. To instantiate our UPSI protocol, we recommend employing the PSU protocol by Zhang et al. [29] due to its linear computational complexity. Furthermore, the use of the OKVS proposed by Bienstock et al. [11] can further reduce communication costs. However, this might come at the expense of increased computational overhead. In fact, using these PSU protocols [26,27,28,30] to instantiate our UPSI protocol is feasible, as they have linear communication costs and do not result in significant performance loss. In fact, we will mention later that instantiating our UPSI using the PSU protocol by Kolesnikov et al. [26] is already very efficient.

4.2 Protocol Construction

We combine all the components mentioned earlier to construct our UPSI protocol, as illustrated in Figure 4. Our UPSI protocol reveals only the size of the addition and deletion sets while enabling intersection computation in a constant number of rounds. Our UPSI protocol is highly robust; even if both parties add elements that already exist or delete elements that do not exist, it will not encounter running fails. We also provide explanations for each phase of the protocol here to aid the reader in understanding.

Initialization. In the initialization phase, $\mathcal{P}_X$ and $\mathcal{P}_Y$ each use X and Y to execute a conventional PSI once and compute $H_X = \{\mathsf{H}(x)^{k_x}\}$ and $H_Y = \{\mathsf{H}(y)^{k_y}\}$ for all $x \in X$ and $y \in Y$. This phase only needs to be executed once, especially the computations of H_X and H_Y. Even if the base PSI is re-executed (when the updated sets are too large and executing UPSI is not as efficient as re-executing the base PSI), there is no need to regenerate H_X and H_Y (only simple updates are required).

Deletion. $\mathcal{P}_X$ and $\mathcal{P}_Y$ use $D_X = \{\mathsf{H}(x^-)^{k_x}\}$ and $D_Y = \{\mathsf{H}(y^-)^{k_y}\}$ for all $x^- \in X^-$ and $y^- \in Y^-$ to update H_X and H_Y, respectively.

Addition. This phase is similar to the previous one, where $\mathcal{P}_X$ and $\mathcal{P}_Y$ use X^+ and Y^+ to update H_X and H_Y, respectively.

Compute the intermediate intersection. $\mathcal{P}_X$ and $\mathcal{P}_Y$ execute our asymmetric PSI protocol, with $\mathcal{P}_X$ obtaining $T = Y^+ \cap X_1$. Similarly, $\mathcal{P}_X$ and $\mathcal{P}_Y$ execute the asymmetric PSI protocol again, with $\mathcal{P}_Y$ obtaining $V = X^+ \cap Y_1$. We would like to emphasize that during this phase, $\mathcal{P}_X$ and $\mathcal{P}_Y$ need to regenerate the secret values k'_x and k'_y, as this prevents both parties from learning more information about X^+ and Y^+ during multiple executions of the UPSI. For example, if $\mathcal{P}_X$ does not regenerate k'_x, then a certain x^+ from $\mathcal{P}_X$ may be added during one update, deleted in a subsequent UPSI, and then added again.

In this case, $\mathcal{P}_Y$ would be able to detect this fact because the mask for this x^+ remains the same. To avoid such leakage, we should ensure that even for the same x^+, its mask is different in each execution of the UPSI.

Compute the intermediate union. In this phase, $\mathcal{P}_X$ and $\mathcal{P}_Y$ need to execute the PSU protocol twice, after which both parties obtain $U' = (X^- \cap I) \cup (Y^- \cap I)$.

Compute the updated intersection. Both parties can locally compute I_1, and this completes the UPSI.

If both parties need to perform the next UPSI, they only need to re-execute all phases, except for the initialization phase, based on the output result I using the new X^+, X^-, Y^+, and Y^-.

4.3 Updatable PSI Correctness Proof

If we focus on just one phase, the correctness of each phase of the UPSI in Figure 4 is trivial. What we need to focus on is whether I_1 is indeed the intersection of X_1 and Y_1. Therefore, proving the correctness of our protocol essentially involves demonstrating that the intersection $I_1 = (I \setminus U') \cup U$ holds for X_1 and Y_1, where $U = T \cup V$, $T = Y^+ \cap X_1$, $V = X^+ \cap Y_1$, and $U' = (X^- \cap I) \cup (Y^- \cap I)$.

Theorem 1. *Let X and Y be sets with intersection $I = X \cap Y$. We let $X_1 = (X \setminus X^-) \cup X^+, Y_1 = (Y \setminus Y^-) \cup Y^+, T = Y^+ \cap X_1, V = X^+ \cap Y_1, U = T \cup V$, and $U' = (X^- \cap I) \cup (Y^- \cap I)$. Then, the updated intersection $I_1 = X_1 \cap Y_1$ satisfies $I_1 = (I \setminus U') \cup U$.*

Proof. We will demonstrate that $I_1 = (I \setminus U') \cup U$ by proving both inclusions:

$$I_1 \subseteq (I \setminus U') \cup U,$$
$$(I \setminus U') \cup U \subseteq I_1.$$

• $I_1 \subseteq (I \setminus U') \cup U$: Let $z \in I_1$. Then $z \in X_1$ and $z \in Y_1$.

Case 1: If $z \in I$, then $z \in X$ and $z \in Y$. Since $z \in X_1$, it must either not be deleted from X (i.e., $z \notin X^-$) or be re-added (i.e., $z \in X^+$). Similarly, $z \in Y_1$ implies $z \notin Y^-$ or $z \in Y^+$. Since $z \in I$ and $z \in I_1$, $z \in I \setminus U'$ holds if $z \notin U'$ (i.e., $z \notin X^-$ and $z \notin Y^-$).

Case 2: If $z \notin I$, then for $z \in X_1 \cap Y_1$ while $z \notin I$, it must be the case that z was added to at least one of the sets: If $z \in Y^+$ and $z \in X_1$, then $z \in T$. If $z \in X^+$ and $z \in Y_1$, then $z \in V$. Therefore, $z \in U = T \cup V$.

Combining both cases, we have $z \in (I \setminus U') \cup U$.

• $(I \setminus U') \cup U \subseteq I_1$: Let $z \in (I \setminus U') \cup U$.

Case 1: If $z \in I \setminus U'$, then $z \in I$ and $z \notin U'$ hold. Therefore, $z \notin X^-$ and $z \notin Y^-$ due to $z \notin U'$, implying $z \in X_1$ and $z \in Y_1$. Hence, $z \in I_1$.

Case 2: If $z \in U$, then $z \in T$ and $z \in V$ due to $U = T \cup V$. If $z \in T = Y^+ \cap X_1$, then $z \in Y^+$ implies $z \in Y_1$, and $z \in X_1$. If $z \in V = X^+ \cap Y_1$, then $z \in X^+$ implies $z \in X_1$, and $z \in Y_1$. Combining both subcases, we have $z \in I_1$.

Since both inclusions hold, we conclude that: $I_1 = (I \setminus U') \cup U$.

4.4 Complexity Analysis

We comprehensively analyze the computational and communication complexities of Protocol 4 here. We conduct a detailed analysis of the complexities at each phase of the UPSI protocol to determine its overall complexity. We would like to emphasize that this analysis does not include the initialization phase, as it comprises a base PSI and preparatory steps that can be implemented using an efficient conventional PSI protocol, such as those in [6,7,8,9,10]. The repeated execution of the remaining phases alone constitutes our UPSI protocol. For the sake of discussion, we assume $|X| = |Y| = N$ and $|X^+| = |Y^+| = |X^-| = |Y^-| = N_u$. Let h denote the cost of hashing, and e denote the cost of exponentiation.

Deletion. We can observe that both parties perform $\mathcal{O}(N_u \cdot (h+e) + N_u \log N)$ computations. This phase incurs no communication overhead.

Addition. The computational complexity of this phase is the same as that of the deletion phase, with no communication overhead.

Compute the intermediate intersection. In step 8, $\mathcal{P}_Y$ and $\mathcal{P}_X$ performs $\mathcal{O}(N_u \cdot (h + 2e))$ and $\mathcal{O}(N_u \cdot e)$ computations, respectively. The communication overhead is $\mathcal{O}(3 \cdot |\mathbb{G}| \cdot N_u)$. Step 9 is identical to Step 8, except that $\mathcal{P}_X$ and $\mathcal{P}_Y$ interchange roles. Therefore, both $\mathcal{P}_X$ and $\mathcal{P}_Y$ perform $\mathcal{O}(N_u \cdot (h + 3e))$ computations in this phase, requiring a total communication cost of $\mathcal{O}(6 \cdot |\mathbb{G}| \cdot N_u)$.

Compute the intermediate union. This phase mainly involves two invocations of the PSU protocol. We can see that the input size for these two PSU protocol invocations is at most N_u in the worst case. Therefore, according to the existing efficient PSU protocols [26,27,28,29,30], the computational complexity and communication complexity of this phase are $\mathcal{O}(N_u)$ (or possibly $\mathcal{O}(N_u \cdot \log N_u)$) and $\mathcal{O}(N_u)$, respectively.

Compute the updated intersection. The computational cost for both parties in this step is $\mathcal{O}(N_u)$, with no communication cost required.

In summary, the computational complexity and communication complexity of our UPSI protocol can be summarized as $\mathcal{O}(N_u \cdot \log N)$ and $\mathcal{O}(N_u)$, respectively. We will also provide experimental evidence to demonstrate that our analysis is correct.

4.5 Updatable PSI Security Proof

It is evident that the sizes of the update sets for each party, i.e., $|X^+|$, $|Y^+|$, $|Y^-|$, and $|X^-|$, are revealed. This might be an unavoidable form of leakage, as it has been a persistent issue in previous UPSI protocols [14,15,18]. Our UPSI protocol relies on the DDH assumption.

Decisional Diffie-Hellman (DDH) Assumption. Let $\mathbb{G}$ be a cyclic group of prime order p with generator g. Let a, b, c be sampled uniformly at random from $\mathbb{Z}_p$. The DDH assumption states that

$$(g^a, g^b, g^{ab}) \approx (g^a, g^b, g^c).$$

Theorem 2. *Let H be a random oracle. The protocol Π^{UPSI} realizes $\mathcal{F}_{\mathsf{UPSI}}$ against a semi-honest adversary under the DDH assumption, assuming the existence of ideal $\mathcal{F}_{\mathsf{PSI}}$ and $\mathcal{F}_{\mathsf{PSU}}$ in the semi-honest model.*

Proof. Since $\mathcal{P}_X$ and $\mathcal{P}_Y$ are symmetric, we only need to prove that no additional information is revealed when either party is corrupted. Without loss of generality, we assume that $\mathcal{P}_X$ is the corrupted party, while $\mathcal{P}_Y$ remains honest. Before Step 8 of the protocol, that is, prior to the "compute the intermediate intersection phase", all operations are local computations except for the invocation of the ideal $\mathcal{F}_{\mathsf{PSI}}$. Thus, simulating this part of the process is trivial. $\mathcal{P}_X$ will receive $A'_Y = \{\mathsf{H}(y^+)^{k'_y}\}$ for all $y^+ \in Y^+$ from $\mathcal{P}_Y$. Using a sequence of hybrid arguments, we show that the corrupted $\mathcal{P}_X$ cannot distinguish the elements in A'_Y from random values in $\mathbb{G}$.

$\mathcal{H}_0$: This is the view of $\mathcal{P}_X$ in the real execution of Π^{UPSI} when it receives A'_Y.

$\mathcal{H}_{1,i}$: For $i \in \{1, \cdots, N_u\}$, the same as $\mathcal{H}_0$ except that we replace $\mathsf{H}(y^+)^{k'_y}$ in A'_Y with random $g_i \in \mathbb{G}$.

$\mathcal{H}_2$: The view of $\mathcal{P}_X$ as output by the simulator when it finishes receiving A'_Y.

We argue that $\mathcal{H}_{1,i-1}$ and $\mathcal{H}_{1,i}$ are indistinguishable to $\mathcal{P}_X$. If any PPT adversary $\mathcal{A}$ can distinguish the two hybrids, we devise a challenger $\mathcal{C}$ who can break the DDH assumption. $\mathcal{C}$ is given (g, g^a, g^b, g^c) and needs to decide whether c is random or $c = ab$. $\mathcal{C}$ can program $\mathsf{H}(\cdot)$ to return g^b on input y^+, and we let $g^a = g^{k'_y}$. $\mathcal{C}$ receives the challenge mask ϵ. Note that $\mathcal{C}$ does not know that ϵ belongs to $\mathcal{H}_{1,i-1}$ or $\mathcal{H}_{1,i}$. $\mathcal{C}$ sends ϵ to $\mathcal{A}$, and then $\mathcal{A}$ determines whether ϵ belongs to $\mathcal{H}_{1,i-1}$ or $\mathcal{H}_{1,i}$. If $c = ab$, the mask $\epsilon = g^c$, otherwise $\epsilon = g_i$ (since g_i is random). If $\mathcal{A}$ judges that ϵ belongs to $\mathcal{H}_{1,i-1}$, then $\mathcal{C}$ outputs that $c = ab$; otherwise outputs that c is random. Therefore, we can see that if $\mathcal{A}$ can distinguish the mask part of two hybrids, then $\mathcal{C}$ can break the DDH assumption with the same probability.

$\mathcal{P}_X$ also receives $E_X = \{{a'_x}^{k_y}\}$ for all $a'_x \in A'_X$, where $A'_X = \{\mathsf{H}(x^+)^{k'_x}\}$. Note that this process in Step 9 is almost identical to Step 8. Additionally, $\mathcal{P}_X$ will receive $H_Y^+ = \{e_y^{{k'_y}^{-1}}\}$ for all $e_y \in E_Y$ from $\mathcal{P}_Y$. The methods used to prove that $\mathcal{P}_X$ cannot distinguish the elements in E_X and H_Y^+ from random values in $\mathbb{G}$ are similar to those for A'_Y, and thus will not be repeated here.

The remaining parts of the protocol consist entirely of local computations, except for the two invocations of the ideal $\mathcal{F}_{\mathsf{PSU}}$. Therefore, the simulation of the remaining parts of the protocol is trivial.

In summary, our UPSI protocol is secure against semi-honest adversaries.

Π^{UPSI}

Initialization:

1. $\mathcal{P}_X$ and $\mathcal{P}_Y$ randomly sample k_x and k_y from $\mathbb{Z}_p^*$, respectively.
2. $\mathcal{P}_X$ and $\mathcal{P}_Y$ locally compute $H_X = \{\mathsf{H}(x)^{k_x}\}$ and $H_Y = \{\mathsf{H}(y)^{k_y}\}$ for all $x \in X$ and $y \in Y$, respectively.
3. $\mathcal{P}_X$ and $\mathcal{P}_Y$ invoke an ideal $\mathcal{F}^{\mathsf{PSI}}$, and both parties receive the intersection $I = X \cap Y$.

Deletion:

4. If $X^- \neq \emptyset$, then $\mathcal{P}_X$ computes $D_X = \{\mathsf{H}(x^-)^{k_x}\}$ for all $x^- \in X^-$, and then updates $H_X = H_X \backslash D_X$.
5. If $Y^- \neq \emptyset$, then $\mathcal{P}_Y$ computes $D_Y = \{\mathsf{H}(y^-)^{k_y}\}$ for all $y^- \in Y^-$, and then updates $H_Y = H_Y \backslash D_Y$.

Addition:

6. If $X^+ \neq \emptyset$, then $\mathcal{P}_X$ computes $A_X = \{\mathsf{H}(x^+)^{k_x}\}$ for all $x^+ \in X^+$, and then updates $H_X = H_X \cup A_X$.
7. If $Y^+ \neq \emptyset$, then $\mathcal{P}_Y$ computes $A_Y = \{\mathsf{H}(y^+)^{k_y}\}$ for all $y^+ \in Y^+$, and then updates $H_Y = H_Y \cap A_Y$.

Compute the intermediate intersection:

8. $\mathcal{P}_Y$ randomly samples k'_y from $\mathbb{Z}_p^*$. Subsequently, $\mathcal{P}_Y$ computes $A'_Y = \{\mathsf{H}(y^+)^{k'_y}\}$ for all $y^+ \in Y^+$ and sends A'_Y to $\mathcal{P}_X$. Then, $\mathcal{P}_X$ computes $E_Y = \{{a'_y}^{k_x}\}$ for all $a'_y \in A'_Y$ and returns it to $\mathcal{P}_Y$. After that, $\mathcal{P}_Y$ computes $H_Y^+ = e_y^{{k'_y}^{-1}}$ for all $e_y \in E_Y$. Finally, $\mathcal{P}_Y$ sends H_Y^+ to $\mathcal{P}_X$ in a random order. $\mathcal{P}_X$ then computes $H_Y^+ \cap H_X$ to obtain $T = Y^+ \cap X_1$.
9. $\mathcal{P}_X$ randomly samples k'_x from $\mathbb{Z}_p^*$. Subsequently, $\mathcal{P}_X$ computes $A'_X = \{\mathsf{H}(x^+)^{k'_x}\}$ for all $x^+ \in X^+$ and sends A'_X to $\mathcal{P}_Y$. Then, $\mathcal{P}_Y$ computes $E_X = \{{a'_x}^{k_y}\}$ for all $a'_x \in A'_X$ and returns it to $\mathcal{P}_X$. After that, $\mathcal{P}_X$ computes $H_X^+ = e_x^{{k'_x}^{-1}}$ for all $e_x \in E_X$. Finally, $\mathcal{P}_X$ sends H_X^+ to $\mathcal{P}_Y$ in a random order. $\mathcal{P}_Y$ then computes $H_X^+ \cap H_Y$ to obtain $V = X^+ \cap Y_1$.

Compute the intermediate union:

10. $\mathcal{P}_X$ and $\mathcal{P}_Y$ invoke an ideal $\mathcal{F}^{\mathsf{PSU}}$, and both parties then receive $U = T \cup V$.
11. $\mathcal{P}_X$ and $\mathcal{P}_Y$ locally compute $X^- \cap I$ and $Y^- \cap I$, respectively.
12. $\mathcal{P}_X$ and $\mathcal{P}_Y$ invoke an ideal $\mathcal{F}^{\mathsf{PSU}}$, and both parties then receive $U' = (X^- \cap I) \cup (Y^- \cap I)$.

Compute the updated intersection:

13. $\mathcal{P}_X$ and $\mathcal{P}_Y$ can locally compute $I_1 = (I \backslash U') \cup U$.

The next UPSI operation:

14. $\mathcal{P}_X$ and $\mathcal{P}_Y$ prepare the new X^+, X^- and Y^+, Y^-, respectively, and redefine the current X_1 as X and the current Y_1 as Y. They then re-execute all the phases above **except for the initialization**.

Fig. 4: Our complete UPSI protocol.

5 Evaluation

In this section, we provide a comprehensive evaluation of our work, including the performance of our UPSI protocol under different network environments and various parameters, a comparison with state-of-the-art protocols, and the threshold for the update set size at which both parties should re-execute the base PSI protocol, considering different network environments and various input sizes. Our implementation is available on GitHub: https://github.com/ShallMate/upsi.

5.1 Experimental Setup

We used two Docker[1] containers on the workstation with Intel(R) Xeon(R) Gold 6230R CPU @ 2.10 GHz, 52 cores, and 128 GB RAM to simulate $\mathcal{P}_X$ and $\mathcal{P}_Y$. The experiment runs on the CentOS system. Our UPSI protocols is implemented using the YACL library[2], which is a C++ library that contains common cryptography, network and I/O modules. We set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$. We evaluate the performance of our protocol under both LAN and WAN settings. We simulate the LAN connection with 0.2 ms RTT network latency and 1 Gbps network bandwidth. We simulate the WAN connection using the Linux "tc" command. For the bandwidth in the WAN setting, we follow the configuration of Badrinarayanan et al. [15] and present our results at 200 Mbps, 50 Mbps, and 5 Mbps with an RTT latency of 80 ms. We use the Four$\mathbb{Q}$ curve [34] to instantiate $\mathbb{G}$ and SHA512 for the hash function H. We use the PSU protocol proposed by Kolesnikov et al. [26] to instantiate our UPSI instead of the linear-computation-cost PSU proposed by Zhang et al. [29], as their provided source code is written in Java[3], which makes it inconvenient for us to integrate with it. Nevertheless, we observe that our protocol remains highly efficient using [26] based on the experimental results. We would like to emphasize that if the PSU protocol by Zhang et al. [29] were used, the performance of our protocol might be even better than what is demonstrated in this paper. For the base PSI used to compute the initial intersection between both parties, we choose the state-of-the-art two-party PSI proposed by Raghuraman et al. [10]. For the sake of presenting the results, we assume $|X| = |Y| = N$ and $|X^+| = |Y^+| = |X^-| = |Y^-| = N_u$.

5.2 The Performance of Our UPSI

We present the specific performance of our UPSI protocol under different values of N and N_u, as well as various network environments. We set the input size $N \in \{2^{19}, 2^{20}, 2^{21}, 2^{22}, 2^{23}, 2^{24}\}$ and the update set size $N_u \in \{2^7, 2^8, 2^9, 2^{10}, 2^{11}, 2^{12}, 2^{13}, 2^{14}, 2^{15}, 2^{16}\}$. In previous works [14,15], the performance was shown only up

[1] https://www.docker.com
[2] https://github.com/secretflow/yacl
[3] https://github.com/alibaba-edu/mpc4j

Table 2: Communication cost (in MB) and running time (in seconds) of the initialization phase under different values of N and various network bandwidths.

N	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
Running Time (s) (LAN, 1 Gbps)	3.06	6.77	13.9	32.9	68.7	147.5
Running Time (s) (WAN, 200 Mbps)	4.88	9.62	19.62	42.32	89.21	189.49
Running Time (s) (WAN, 50 Mbps)	8.69	17.36	35.12	73.15	150.95	312.59
Running Time (s) (WAN, 5 Mbps)	55.62	110.74	221.39	444.54	894.31	1799.39
Comm. (MB)	26.1	51.9	103.4	206.6	412.9	825.6

to $N = 2^{22}$ and $N_u = 2^{12}$. We provide as many parameters as possible to give readers a better understanding of the performance of our protocol.

First, we present the costs of our UPSI protocol during the initialization phase. Essentially, this is a phase of the pre-computation for our UPSI protocol and only needs to be executed once. The costs of this phase only depend on N and are linear with respect to it. This phase primarily requires a single execution of the base PSI protocol, as well as pre-computation by both parties for the masks used in the subsequent asymmetric PSI. We present the communication cost (in MB) and running time (in seconds) of the initialization phase of our UPSI under different values of N and various network environments in Table 2. We can see that when $N = 2^{24}$, both parties require 147.5 seconds in the LAN setting and 825.6 MB of communication to complete the initialization phase. However, since the initialization phase requires executing the base PSI, it can become relatively slow when the input size is large and the bandwidth is low due to the significant communication overhead it incurs. For example, when $N = 2^{24}$ and the bandwidth is 5 Mbps, the initialization phase takes nearly half an hour (1799.39 seconds) to complete. Fortunately, this phase is essentially a setup phase and only needs to be executed once. Therefore, we do not need to worry too much about the costs in this context. We do not include the costs of the initialization phase in the subsequent UPSI evaluations. Note that the communication cost for the base PSI here is slightly higher than executing a one-way PSI protocol by Raghuraman et al. [10] alone, as the party receiving the intersection needs to send it to the other party.

Next, in Table 3, we present the communication cost (in MB) and running time (in seconds) of our protocol under different values of N and N_u and various network environments. Since our UPSI protocol enjoys extremely low communication costs, it can quickly obtain the updated intersection even under very low bandwidth conditions. For example, with a bandwidth of 5 Mbps, both parties can complete the UPSI protocol in 4.05 seconds when $N = 2^{20}$ and $N_u = 2^8$. When dealing with large-scale data, our protocol can also quickly complete the computation of the updated intersection. For example, when $N = 2^{24}$ and $N_u = 2^{12}$, our UPSI protocol completes the intersection computation in 17.91, 19.27, 22.16, and 56.87 seconds under LAN, 200 Mbps, 50 Mbps, and 5

Mbps WAN settings, respectively. Moreover, our protocol remains efficient even when the size of the update sets is relatively large. Specifically, when $N = 2^{24}$ and $N_u = 2^{16}$, our protocol requires only 307.87, 144.67, 160.47, 206.65, and 760.81 seconds under LAN, 500 Mbps, 50 Mbps, and 5 Mbps WAN settings, respectively. Therefore, it can be observed that our protocol is efficient even with large-scale data, under low bandwidth conditions, and with relatively large update set sizes.

We examine the experimental results in Table 4 to see if the computational complexity and communication complexity analyzed in Section 4.4 are correct. It can be observed that when N_u is constant, changing N does not affect the communication overhead of our UPSI protocol; when N is constant, the communication overhead of our UPSI increases linearly with the growth of N_u. Regarding computational complexity, we can see that when N_u is constant, the running time in the LAN gradually increases as N grows. For example, when $N_u = 2^{10}$, the running times of our UPSI protocol are 2.45 seconds and 2.75 seconds for $N = 2^{19}$ and $N = 2^{20}$, respectively. Additionally, the running time of our UPSI protocol also grows linearly with N_u. For example, when $N = 2^{24}$, the running times of our UPSI protocol are 76.02 and 144.67 seconds for $N_u = 2^{15}$ and $N_u = 2^{16}$, respectively. Therefore, the computational complexity and communication complexity of our UPSI protocol are indeed $\mathcal{O}(N_u \cdot \log N)$ and $\mathcal{O}(N_u)$, respectively.

5.3 Comparison with State-of-The-Art Protocols

We compare our UPSI protocol with the state-of-the-art conventional PSI protocol [10] and UPSI protocol [15]. In [15], there is also a UPSI protocol that only supports addition operations, which we have included in the comparison to demonstrate the efficiency of our protocol. We present the communication cost (in MB) and running time (in seconds) of our protocol in comparison with [10,15] in Table 4.

We summarize our experimental results in terms of communication improvement, computation improvement, and end-to-end running time.

Communication Improvement. Our protocol outperforms RR22 [10] by $10 \sim 2066\times$ in all settings. When $N = 2^{22}$ and $N_u = 2^4$, the communication cost of RR22 is 206.65 MB, whereas our protocol requires only 0.1 MB. Additionally, our protocol achieves a $5 \sim 6\times$ reduction compared to the version that supports only addition in [15]. When $N = 2^{22}$ and $N_u = 2^{10}$, it requires 31.5 MB of communication, whereas our UPSI needs only 4.88 MB. Our protocol reduces communication overhead by $587 \sim 755\times$ compared to the state-of-the-art UPSI protocol [15] that supports both addition and deletion. When $N = 2^{20}$ and $N_u = 2^{10}$, it requires 3687 MB of communication, whereas our UPSI still needs only 4.88 MB. It can be observe that our protocol enjoys extremely low communication cost compared to existing works. Therefore, our UPSI protocol has a significant advantage in terms of communication overhead.

Computation Improvement. Our protocol achieves up to a $32\times$ reduction in computation overhead compared to the version that supports only addition in [15]. Specifically, when $N = 2^{20}$ and $N_u = 2^{10}$, it takes 87.6 seconds to complete the protocol, whereas our UPSI protocol requires only 2.75 seconds in the LAN setting. Furthermore, our protocol reduces computation overhead by $194 \sim 2015\times$ compared to the state-of-the-art UPSI protocol [15] that supports both addition and deletion. For example, when $N = 2^{20}$ and $N_u = 2^{10}$, it takes 5543.5 seconds to complete the protocol, whereas our UPSI protocol still requires only 2.75 seconds. It is evident that our UPSI protocol also has a significant advantage in terms of computation overhead compared to the conventional PSI [15].

End-to-End. The end-to-end running time of our protocol begins to outperform the version that supports only addition in [15] under the vast majority of parameters, except when $N_u \leq 2^4$ for $N = 2^{22}$ and the bandwidth is greater than 5 Mbps. In these settings, our protocol can be $3 \sim 32\times$ faster than the latter. For example, when $N = 2^{20}$ and $N_u = 2^{10}$, the latter requires 87.6, 89.48, 93.92, and 147.22 seconds to complete the protocol under LAN, 200 Mbps, 50 Mbps, and 5 Mbps WAN settings, respectively, whereas our protocol only needs 2.75, 3.4, 4.13, and 12.91 seconds under the same bandwidths. In comparison with the version that supports both addition and deletion in [15], our UPSI demonstrates an advantage in the end-to-end running time across all settings. Our protocol can be up to $1684\times$ faster than the state-of-the-art UPSI protocol [15] that supports both addition and deletion when the bandwidth is 200 Mbps. Specifically, when $N = 2^{20}$ and $N_u = 2^{10}$, it takes 5727.75 seconds to complete the intersection computation, whereas our UPSI requires only 3.4 seconds with a bandwidth of 200 Mbps. Furthermore, our UPSI also has an advantage in overall running time compared to state-of-the-art conventional PSI [10] across all settings. When the bandwidth is 5 Mbps, our protocol can be $8 \sim 176\times$ faster than [10]. Specifically, when $N = 2^{22}$ and $N_u = 2^{10}$, [10] takes 420.57 seconds to complete the intersection computation, whereas our UPSI requires only 14.83 seconds with a bandwidth of 5 Mbps. Therefore, we can see that our protocol also has an advantage in end-to-end running time due to its extremely low communication.

In summary, our protocol has significant advantages over the current optimal protocols in terms of communication improvement, computation improvement, and end-to-end running time under different bandwidths, achieving up to three orders of magnitude improvement.

5.4 The Threshold of N_u

In many real-world applications, the updated sets can be small compared to the entire sets. However, determining the threshold at which N_u grows large enough that running UPSI becomes less efficient than executing a conventional PSI with updated inputs is crucial, and previous works [14,15,18] seem to have overlooked this aspect. We experiment with different values of N and N_u to evaluate the threshold of N_u at which our UPSI protocol becomes less efficient than directly re-executing the state-of-the-art conventional two-party PSI protocol [10]. We consider this a crucial experiment, as there may indeed be a threshold value of N_u beyond which executing the UPSI protocol could be less efficient than re-running a conventional two-party PSI.

As shown in Figure 5, we present a comparison of our UPSI protocol with [10] under different bandwidths and updated set sizes when $N \in \{2^{19}, 2^{20}, 2^{21}, 2^{22}, 2^{23}, 2^{24}\}$. From Figures 5a to 5f, we can see that the performance of our protocol under different N shows a consistent increasing trend as N_u increases. Let N_u^t be a threshold for N_t, beyond which it becomes more efficient to re-run the state-of-the-art conventional PSI [10] rather than executing our UPSI protocol. We can determine the different values of N_u^t under various bandwidths by observing when the running time of our UPSI exceeds that of the conventional PSI [10] under the same bandwidth. We summarize the threshold N_u^t results for different values of N and various bandwidths in Table 5 for the convenience of readers. After knowing the value of N_u^t, if both parties need to update a set larger than N_u^t, they should re-execute the conventional PSI instead of continuing with the proposed UPSI. For example, when $N = 2^{22}$, as long as $|N^+| \leq 2^{15}$ and $|N^-| \leq 2^{15}$, executing our UPSI protocol with a bandwidth of 5 Mbps will definitely be faster than executing the conventional PSI [10]. We would like to emphasize that although 2^{15} still has a significant gap compared to 2^{22}, this is an impressive result compared to previous works. Specifically, the current state-of-the-art UPSI [15] that supports addition and deletion has an actual N_u^t of only 2^4 under the same parameters. In other words, we have expanded this threshold by at least 2^{11} (2048) times in this setting. Even in the LAN setting, the N_u^t for this configuration is 2^{11}, while [15] is only efficient compared to [10] under very low bandwidth and small update set sizes. Moreover, we can observe that as N increases, N_u^t also tends to increase in a certain pattern. For example, when N is fixed, the value of N_u^t is approximately $N/2^7$ under a bandwidth of 5 Mbps. Since N_u is generally much smaller than N, our protocol is sufficient to handle most cases, meaning it is faster than conventional PSI in most scenarios where both parties update their sets.

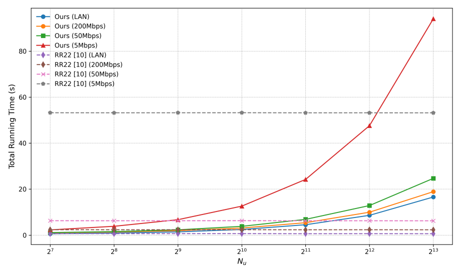
Therefore, it is clear that our protocol indeed addresses the problem we posed in the introduction. We have successfully constructed a UPSI that supports arbitrary additions and deletions of elements while ensuring faster performance than re-executing a conventional PSI in most cases, rather than being limited to highly specific parameters/bandwidths.

Table 3: Communication cost (in MB) and running time (in seconds) of our protocol under different values of N and N_u, and various network environments.

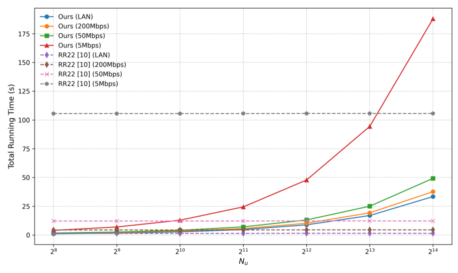
N	N_u	Comm. (MB)	Total Running Time (s)			
			LAN	200Mbps	50Mbps	5Mbps
2^{19}	2^7	0.65	0.58	1.01	1.11	2.28
	2^8	1.28	0.89	1.35	1.54	3.84
	2^9	2.45	1.41	1.93	2.3	6.71
	2^{10}	4.88	2.45	3.09	3.82	12.6
	2^{11}	9.65	4.52	5.4	6.85	24.22
2^{20}	2^8	1.28	1.09	1.56	1.75	4.05
	2^9	2.45	1.7	2.22	2.59	7.0
	2^{10}	4.88	2.75	3.4	4.13	12.91
	2^{11}	9.65	4.76	5.64	7.09	24.47
	2^{12}	19.28	8.85	10.21	13.11	47.81
2^{21}	2^9	2.45	2.33	2.85	3.22	7.63
	2^{10}	4.88	3.42	4.06	4.79	13.57
	2^{11}	9.65	5.54	6.42	7.87	25.25
	2^{12}	19.28	9.52	10.89	13.78	48.49
	2^{13}	38.54	17.55	19.88	25.66	95.02
2^{22}	2^{10}	4.88	4.67	5.31	6.05	14.83
	2^{11}	9.65	6.97	7.85	9.3	26.67
	2^{12}	19.28	10.81	12.17	15.07	49.77
	2^{13}	38.54	19.47	21.79	27.57	96.94
	2^{14}	77.04	35.0	39.25	50.81	189.48
2^{23}	2^{11}	9.65	9.42	10.3	11.75	29.12
	2^{12}	19.28	14.16	15.53	18.42	53.13
	2^{13}	38.54	21.18	23.51	29.29	98.65
	2^{14}	77.04	37.75	42.01	53.56	192.24
	2^{15}	153.99	70.5	78.6	101.7	378.87
2^{24}	2^{12}	19.28	17.91	19.27	22.16	56.87
	2^{13}	38.54	27.42	29.75	35.53	104.89
	2^{14}	77.04	44.06	48.31	59.87	198.55
	2^{15}	153.99	76.02	84.12	107.22	384.39
	2^{16}	307.87	144.67	160.47	206.65	760.81

Table 4: Communication cost (in MB) and running time (in seconds) of our protocol in comparison with prior work. If the existing work is optimal, we highlight it in green; if our work is optimal, we highlight it in red. We did not record experimental results with a running time exceeding two hours.

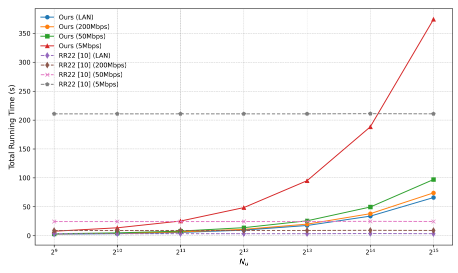
N	N_u	Protocol	Comm. (MB)	Total Running Time (s)			
				LAN	200Mbps	50Mbps	5Mbps
2^{20}	-	RR22 [10]	51.88	1.41	4.4	12.19	105.57
	2^4	BMS$^+$24 [15] (addition-only)	0.50	1.52	1.95	2.02	2.52
	2^6		1.95	5.49	5.99	6.28	9.79
	2^8		7.57	22.8	23.58	24.71	38.34
	2^{10}		29.61	87.6	89.48	93.92	147.22
	2^4	BMS$^+$24 [15] (addition & deletion)	58.7	101.2	104.54	113.34	219.0
	2^6		231	363.3	375.25	409.9	825.7
	2^8		922	1398.4	1445.25	1584.6	3256.8
	2^{10}		3687	5543.5	5727.75	6280.8	-
	2^4	Ours	0.10	0.52	0.93	0.95	1.13
	2^6		0.34	0.66	1.08	1.13	1.74
	2^8		1.28	1.09	1.56	1.75	4.05
	2^{10}		4.88	2.75	3.4	4.13	12.91
2^{22}	-	RR22 [10]	206.65	8.17	18.87	49.77	420.57
	2^4	BMS$^+$24 [15] (addition-only)	0.53	1.62	2.05	2.13	3.08
	2^6		2.06	5.45	5.95	6.26	9.97
	2^8		8.03	23.6	24.4	25.61	40.06
	2^{10}		31.5	92.4	94.38	99.1	155.8
	2^4	BMS$^+$24 [15] (addition & deletion)	61.6	108.3	111.78	121.02	231.9
	2^6		243	402	414.55	451.0	888.4
	2^8		927	1603.7	1650.45	1789.5	3458.1
	2^{10}		-	-	-	-	-
	2^4	Ours	0.10	1.77	2.18	2.2	2.39
	2^6		0.34	2.41	2.83	2.88	3.49
	2^8		1.28	2.69	3.16	3.35	5.65
	2^{10}		4.88	4.70	5.31	6.05	14.83



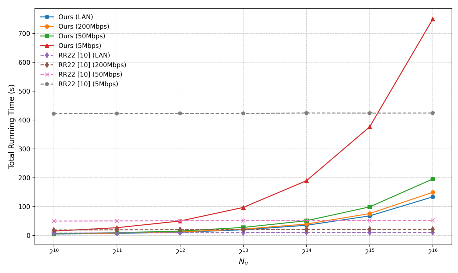
(a) $N = 2^{19}$



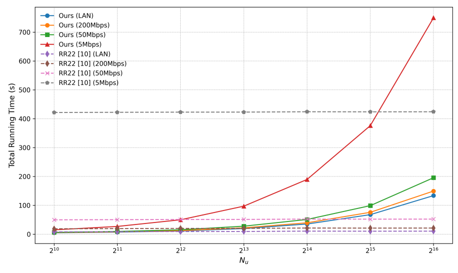
(b) $N = 2^{20}$



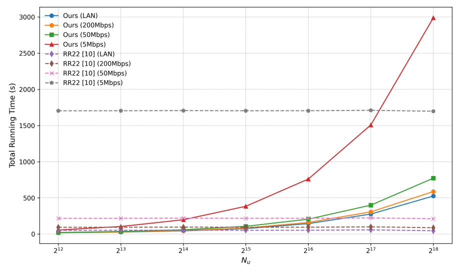
(c) $N = 2^{21}$



(d) $N = 2^{22}$



(e) $N = 2^{23}$



(f) $N = 2^{24}$

Fig. 5: Comparison with the state-of-the-art conventional PSI [10] under different bandwidths and updated set sizes when $N \in \{2^{19}, 2^{20}, 2^{21}, 2^{22}, 2^{23}, 2^{24}\}$.

Table 5: The value of N_u^t for different N under various bandwidths, where N_u^t is a threshold for N_t, beyond which it becomes more efficient to re-run the state-of-the-art conventional PSI [10] rather than executing our UPSI protocol.

N	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
N_u^t (LAN, 1 Gbps)	2^7	2^8	2^9	2^{11}	2^{13}	2^{14}
N_u^t (WAN, 200 Mbps)	2^9	2^{10}	2^{11}	2^{12}	2^{14}	2^{15}
N_t^t (WAN, 50 Mbps)	2^{10}	2^{11}	2^{12}	2^{14}	2^{15}	2^{16}
N_t^t (WAN, 5 Mbps)	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}

6 Limitations and Discussion

This section primarily discusses some limitations of our UPSI protocol that we are currently aware of and outlines our future work.

First, our UPSI protocol is secure in the semi-honest model but does not provide security against malicious adversaries. To the best of our knowledge, no maliciously secure UPSI protocol has been proposed so far. Therefore, we leave the construction of a maliciously secure UPSI protocol as an open problem and a direction for our future work.

Secondly, our UPSI protocol is a fair UPSI, meaning that both parties obtain the intersection. Therefore, our work may not be suitable for PSI scenarios where only one party needs to obtain the intersection. However, this does not imply that our work is without value. For example, in vertical federated learning [13], the first step is usually to use a PSI protocol for sample alignment, where all parties typically need to obtain the intersection. In the next section, we also provide some application scenarios for the UPSI protocol proposed in this paper to illustrate its strong practical significance. However, we still want to improve our UPSI protocol to allow only one party to receive the intersection, which is currently included in our future work.

We plan to address the above two limitations in future work and welcome further research aimed at improving our UPSI protocol.

7 Applications

In this section, we present three application scenarios for our UPSI protocol to illustrate its practical role.

Vertical Federated Learning. Vertical federated learning [13] is a highly favored approach for joint model training in the industry. It refers to multiple companies possessing different feature spaces for the same set of samples, aiming to improve model accuracy through feature expansion. The prerequisite for vertical federated learning is achieving privacy-preserving entity alignment, which involves identifying the common sample IDs across all companies. In fact, this task is typically accomplished using a PSI protocol. However, in practice, data is not stable and actually requires continuous updates. Here we cite a passage from Meta's paper [49]: "*But a typical scenario is for one party's dataset of records to be large and stable for some time, while the other party's dataset arrives in a streaming fashion and in small batches. For example, parameters of a machine learning model can be continuously updated as new batches of records arrive.*" This means that if we use a conventional PSI protocol, we would need to execute it multiple times to continually perform privacy-preserving entity alignment. Although they also use a PSI variant called streaming PSI to attempt to address this issue, this construction is not efficient due to the expensive homomorphic encryption involved (taking up to about two hours to complete the intersection computation for input sizes less than 2^{24} on a c5.18xlarge AWS instance). As shown in Table 3, with our protocol, the entire computation can be completed in less than 20 seconds for an update size of 2^{12} (4096), even taking only around 56 seconds under a 5 Mbps bandwidth. Furthermore, in the vertical federated learning scenario, all participating parties need to obtain the intersection (otherwise, they cannot know which samples are involved in model training). Thus, our UPSI protocol is well-suited for vertical federated learning.

Medical Data Sharing. PSI is also frequently used for privacy-preserving medical data sharing [50,51]. At the same time, we know that medical data, such as epidemic monitoring and case tracking, requires frequent data updates. For example, during the COVID-19 pandemic, the number of new hospital cases and test results can change rapidly. The updatable feature of the UPSI protocol enables hospitals to quickly add the latest data to the intersection, assisting relevant departments and hospitals in obtaining real-time analyses. Furthermore, the capability for dynamic updates also facilitates long-term collaborative case analysis. Suppose a hospital identifies a new case or updates test data. In that case, it can swiftly integrate the new information into the intersection via the UPSI protocol, ensuring that all collaborating parties receive the most timely information. If multiple hospitals identify cases of the same patient with a specific disease, the parties can conduct in-depth analyses based on the intersection data to explore information such as causes and treatment options. This cross-hospital collaboration facilitates knowledge sharing and enhances overall diagnostic and treatment outcomes. Due to the considerable performance advantages of our UPSI compared to previous works, it can assist hospitals in quickly performing these frequently updated intersection computations.

Social Network Analysis. PSI has been widely used in social networks [52,53,54]. Social platforms can identify overlapping users and update their social graphs without disclosing user privacy, thereby providing users with a richer social experience and cross-platform services. When building user social graphs across multiple social platforms, each platform has a large user base, making it relatively expensive to perform a PSI protocol. However, the friend lists and interests of these users may change frequently. Therefore, using a conventional PSI protocol would result in significant resource waste. Each platform can use UPSI to address this issue. Moreover, since each platform needs to update its own social graph, they all require to obtain the intersection. As a result, our UPSI protocol is a viable solution worth considering for this scenario, as it demonstrates highly efficient performance compared to previous works in a setting that requires updating sets.

In fact, the application scenarios for our UPSI protocol are not limited to the three we have listed. Since data inherently requires constant updates, the usefulness of our UPSI protocol becomes increasingly evident.

8 Conclusion

In this work, we construct a UPSI protocol that supports arbitrary additions and deletions of elements, offering faster performance compared to re-executing a conventional PSI in most cases without being restricted to highly specific parameters/bandwidths. Our UPSI protocol can be proven secure in the semi-honest model. Experimental results demonstrate that our UPSI protocol is substantially more efficient than the existing state-of-the-art UPSI protocol and the most efficient conventional PSI protocol, achieving two to three orders of magnitude improvements in both computational and communication costs. We also point out some limitations of our UPSI protocol for further research. Finally, we also provide some applications of the proposed UPSI protocol to demonstrate how our protocol can play a significant role in practical scenarios.

9 Acknowledge

This work was supported by the National Key Research and Development Program of China under Grant 2023YFB3106501.

References

1. Atsuko Miyaji, Kazuhisa Nakasho, and Shohei Nishida. Privacy-preserving integration of medical data - A practical multiparty private set intersection. *J. Medical Syst.*, 41(3):37:1–37:10, 2017.
2. Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018.
3. Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 1447–1464, 2019.
4. Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 797–812, 2014.
5. Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pages 515–530. USENIX Association, 2015.
6. Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 818–829. ACM, 2016.
7. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 401–431. Springer, 2019.
8. Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In *Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 34–63. Springer, 2020.

9. Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 901–930. Springer, 2021.
10. Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In *ACM SIGSAC Conference on Computer and Communications Security, Los Angeles, CA, USA, November 7-11, 2022*, pages 2505–2517. ACM, 2022.
11. Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Near-optimal oblivious key-value stores for efficient psi, PSU and volume-hiding multi-maps. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 301–318. USENIX Association, 2023.
12. Guowei Ling, Fei Tang, Chaochao Cai, Jinyong Shan, Haiyang Xue, Wulu Li, Peng Tang, Xinyi Huang, and Weidong Qiu. P^{2}frpsi: Privacy-preserving feature retrieved private set intersection. *IEEE Trans. Inf. Forensics Secur.*, 19:2201–2216, 2024.
13. Yang Liu, Yan Kang, Tianyuan Zou, Yanhong Pu, Yuanqin He, Xiaozhou Ye, Ye Ouyang, Ya-Qin Zhang, and Qiang Yang. Vertical federated learning: Concepts, advances, and challenges. *IEEE Trans. Knowl. Data Eng.*, 36(7):3615–3634, 2024.
14. Saikrishna Badrinarayanan, Peihan Miao, and Tiancheng Xie. Updatable private set intersection. *Proc. Priv. Enhancing Technol.*, 2022(2):378–406, 2022.
15. Saikrishna Badrinarayanan, Peihan Miao, Xinyi Shi, Max Tromanhauser, and Ruida Zeng. Updatable private set intersection revisited: Extended functionalities, deletion, and worst-case complexity. *Cryptology ePrint Archive*, 2024.
16. Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, pages 577–594, 2010.
17. Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*, pages 79–88, 2006.
18. Archita Agarwal, David Cash, Marilyn George, Seny Kamara, Tarik Moataz, and Jaspal Singh. Updatable private set intersection from structured encryption. *IACR Cryptol. ePrint Arch.*, page 1183, 2024.
19. Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1243–1255, 2017.
20. Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1223–1237, 2018.
21. Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 1135–1150, 2021.

22. Rasoul Akhavan Mahdavi, Nils Lukas, Faezeh Ebrahimianghazani, Thomas Humphries, Bailey Kacsmar, John A. Premkumar, Xinda Li, Simon Oya, Ehsan Amjadian, and Florian Kerschbaum. PEPSI: practically efficient private set intersection in the unbalanced setting. In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, 2024.
23. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106, 2011.
24. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325, 2012.
25. Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, pages 121–133, 2001.
26. Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part II*, pages 636–666, 2019.
27. Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private set operations from oblivious switching. In *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part II*, pages 591–617, 2021.
28. Yanxue Jia, Shifeng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. Shuffle-based private set union: Faster and more secure. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 2947–2964, 2022.
29. Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Linear private set union from multi-query reverse private membership test. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 337–354, 2023.
30. Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, and Dawu Gu. Scalable private set union, with stronger security. In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, 2024.
31. Catherine A. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *IEEE Symposium on Security and Privacy*, pages 134–137, 1986.
32. Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 78–86. ACM, 1999.
33. Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, pages 207–228, 2006.
34. Craig Costello and Patrick Longa. Fourıı: Four-dimensional decompositions on a ıı-curve over the mersenne prime. In *Advances in Cryptology - ASIACRYPT 2015*

- 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I, pages 214–235, 2015.

35. Mike Rosulek and Ni Trieu. Compact and malicious private set intersection for small sets. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 1166–1181. ACM, 2021.
36. Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In Radu Sion, editor, *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010, Revised Selected Papers*, volume 6052 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2010.
37. Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 370–389. IEEE, 2020.
38. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2003.
39. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from paxos: Fast, malicious private set intersection. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 739–767. Springer, 2020.
40. Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, pages 121–133, 2001.
41. Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 395–425. Springer, 2021.
42. Phillipp Schoppmann, Adrià Gascón, Mariana Raykova, and Benny Pinkas. Make some ROOM for the zeros: Data sparsity in secure distributed machine learning. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 1335–1350. ACM, 2019.
43. Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 502–534. Springer, 2021.

44. Fei Tang, Guowei Ling, Chaochao Cai, et al. Solving small exponential ECDLP in ec-based additively homomorphic encryption and applications. *IEEE Trans. Inf. Forensics Secur.*, 18:3517–3530, 2023.
45. Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 557–574, 2013.
46. Aydin Abadi, Changyu Dong, Steven J. Murdoch, and Sotirios Terzis. Multi-party updatable delegated private set intersection. In *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, pages 100–119, 2022.
47. Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptol.*, 13(1):143–202, 2000.
48. Nick Angelou, Ayoub Benaissa, Bogdan Cebere, William Clark, Adam James Hall, Michael A. Hoeh, Daniel Liu, Pavlos Papadopoulos, Robin Roehm, Robert Sandmann, Phillipp Schoppmann, and Tom Titcombe. Asymmetric private set intersection with applications to contact tracing and private vertical federated machine learning. *CoRR*, abs/2011.09350, 2020.
49. Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, et al. Private matching for compute. *IACR Cryptol. ePrint Arch.*, page 599, 2020.
50. Yalian Qian, Jian Shen, Pandi Vijayakumar, and Pradip Kumar Sharma. Profile matching for iomt: A verifiable private set intersection scheme. *IEEE J. Biomed. Health Informatics*, 25(10):3794–3803, 2021.
51. Mohammed Ramadan and Shahid Raza. Secure equality test technique using identity-based signcryption for telemedicine systems. *IEEE Internet Things J.*, 10(18):16594–16604, 2023.
52. Ghita Mezzour, Adrian Perrig, Virgil D. Gligor, and Panos Papadimitratos. Privacy-preserving relationship path discovery in social networks. In *Cryptology and Network Security, 8th International Conference, CANS 2009, Kanazawa, Japan, December 12-14, 2009. Proceedings*, pages 189–208, 2009.
53. Pili Hu, Sherman S. M. Chow, and Wing Cheong Lau. Secure friend discovery via privacy-preserving and decentralized community detection. *CoRR*, abs/1405.4951, 2014.
54. Jingwei Hu, Yongjun Zhao, Benjamin Hong Meng Tan, Khin Mi Mi Aung, and Huaxiong Wang. Enabling threshold functionality for private set intersection protocols in cloud computing. *IEEE Trans. Inf. Forensics Secur.*, 19:6184–6196, 2024.