Efficient Updatable PSI from Asymmetric PSI and PSU

Guowei Ling, Peng Tang, and Weidong Qiu

School of Cyber Science and Engineering, Shanghai Jiao Tong University, Shanghai, 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. 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. This work combines asymmetric PSI with Private Set Union (PSU) to present a novel UPSI protocol, which supports arbitrary additions and deletions of elements, offering a flexible approach to update sets. Furthermore, our protocol enjoys efficient performance compared to previous work. Specifically, we implement our protocol and compare it against state-of-the-art conventional PSI and UPSI protocols. Experimental results demonstrate that our UPSI protocol achieves up to three orders of magnitude reduction in computational overhead and incurs $190 \sim 707\times$ less communication overhead than the state-of-the-art UPSI protocol that supports arbitrary additions and deletions. Our implementation is available at: https://github.com/ShallMate/upsi.

1 Introduction

Private Set Intersection (PSI) 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 many efficient PSI protocols having been proposed [4–8]. The most efficient PSI [8] can compute the intersection in about one second for input sizes around one million, requiring only tens of megabytes of communication.

Despite the significant performance breakthroughs achieved by these efficient PSI protocols, they are restricted to a *static* setting. That is, 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 some practical applications, the sets are often subject to continuous updates, and the intersection is computed

multiple times as the sets grow or shrink over time. For example, a typical application of PSI is sample alignment before vertical federated learning [9], 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, it would significantly waste resources.

A recent work by Badrinarayanan et al. [10] initiated 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 protocol only supports adding elements, while deletions are implemented through a periodic refresh mechanism, referred to as "weak deletion." Very recently, Badrinarayanan et al. [11] proposed a revised version, which supports adding and deleting elements. However, from the experimental results in Table 4 of [11], this protocol introduces new challenges: expensive computational and communication overhead. Specifically, it only outperforms a re-execution of the efficient PSI protocol [8] 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 [11] that supports deletion will take more time than re-executing the conventional PSI [8]. 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 new UPSI protocol. To 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 experiments, computation, communication, and end-to-end.

• **Experiments**. We provide a comprehensive report on the performance of our protocol under various input sizes, update set sizes, and bandwidth conditions. Moreover, we also compare our protocol with the state-of-the-art UPSI [11] and conventional PSI [8]. 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 [10, 11].

• **Computation**. Our UPSI protocol reduces computation overhead by $255 \sim 1061\times$ compared to [11] that supports both addition and deletion and most $16\times$ faster compared to the version that supports only addition.

• **Communication**. Our protocol reduces communication overhead by a factor of $190 \sim 707$ compared to [11] that supports both addition and deletion.

Additionally, it achieves a $2 \sim 6\times$ reduction compared to the version that supports only addition. Additionally, our protocol has a communication overhead that is $8 \sim 42\times$ less than that of the conventional PSI protocol [8] 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 $974\times$ faster than the state-of-the-art UPSI protocol [11] that supports both addition and deletion with a bandwidth of 200 Mbps. Furthermore, our protocol can be $5 \sim 24\times$ faster than the state-of-the-art conventional PSI protocol [8] with a bandwidth of 5 Mbps.

1.2 Technical Overview

Our UPSI is inspired by the UPSI framework based on structured encryption [12] proposed by Agarwal et al. [13]. 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. Furthermore, the UPSI protocol [13] provides only a complexity analysis, with neither experimental results nor code implementation available, leaving its concrete performance unclear. Let us introduce the core content of our UPSI protocol step by step. Note that certain specific computational steps are omitted here (but are required in the actual protocol) to maintain clarity.

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 PSI protocol as a base protocol, such as [5–8], to obtain the intersection $I = X \cap Y$. Let $X_0 = X$, $Y_0 = Y$, and $I_0 = I$. The execution process of the i-th UPSI is as follows.

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

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

Compute the intermediate intersection. Let the updated sets of $\mathcal{P}_X$ and $\mathcal{P}_Y$ be denoted as X_i and Y_i, respectively, where $X_i = (X_{i-1} \setminus X_i^-) \cup X_i^+$ and $Y_i = (Y_{i-1} \setminus Y_i^-) \cup Y_i^+$. $\mathcal{P}_X$ and $\mathcal{P}_Y$ execute the asymmetric PSI protocol using X_i and Y_i^+, with $\mathcal{P}_Y$ obtaining $T_i = Y_i^+ \cap X_i$. Similarly, $\mathcal{P}_X$ and $\mathcal{P}_Y$ execute the asymmetric PSI protocol again using X_i^+ and Y_i, with $\mathcal{P}_X$ obtaining $V_i = X_i^+ \cap Y_i$.

Compute the intermediate union. $\mathcal{P}_X$ and $\mathcal{P}_Y$ use V_i and T_i as inputs, respectively, and invoke the PSU protocol, with both parties obtaining $U_i = T_i \cup V_i$. Subsequently, $\mathcal{P}_X$ and $\mathcal{P}_Y$ locally compute $X_i^- \cap I_{i-1}$ and $Y_i^- \cap I_{i-1}$, respectively. Both parties invoke the PSU protocol again, and each receives $U_i' = (X_i^- \cap I_{i-1}) \cup (Y_i^- \cap I_{i-1})$.

Compute the updated intersection. Finally, each party can locally compute $I_i = (I_{i-1} \setminus U_i') \cup U_i$ as the result of $X_i \cap Y_i$.

Our UPSI protocol requires executing a conventional PSI protocol as the base PSI in the initialization phase. After both parties update their sets, the UPSI protocol can be executed repeatedly by following the same steps (except for initialization).

Remark. Our UPSI protocol is a fair UPSI, meaning that both parties obtain the intersection. Therefore, our work may not be suitable for scenarios where

only one party obtains the intersection. However, this does not imply that our protocol is without value. In Appendix A, we provide some applications for our UPSI protocol.

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 work, while UPSI is the goal we aim to achieve.

PSI. Pinkas et al. [4] constructed an efficient PSI protocol based on the Oblivious Transfer (OT) extension [14]. Since then, many efficient PSI protocols [5, 6] have been proposed. However, compared to DH-based PSI protocols [15, 16], these protocols sacrifice communication efficiency in exchange for higher computational efficiency. Fortunately, this changed with the advent of OKVS [17], which provides a convenient way to represent private sets and facilitates subsequent intersection calculations. The communication overhead of the first OKVS-based PSI protocols was high. Garimella et al. [18] addressed this issue, and Rindal et al. [7] subsequently combined OKVS with a VOLE protocol [19] to create an efficient PSI protocol with very low communication. Shortly afterwards, Raghuraman et al. [8] improved the OKVS in [7] and combined it with a more efficient VOLE protocol [20], resulting in a state-of-the-art PSI protocol.

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. The current asymmetric PSI protocols are primarily constructed based on fully homomorphic encryption [21]. Chen et al. [22] introduced optimizations to reduce the multiplicative depth of the function evaluated homomorphically, thereby enhancing efficiency. Furthermore, Chen et al. [23], and Cong et al. [24] employ a combination of OPRF and fully homomorphic encryption, building upon [22].

PSU. We primarily focus on OT-based PSU protocols due to their high efficiency. Kolesnikov et al. [25] 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. [26] proposed a new PSU protocol based on oblivious switching. Jia et al. [27] also proposed two shuffle-based PSU protocols built on the oblivious switching, called the Permute + Share subprotocol. Consequently, the performance of their protocols is similar to that of [26]. Recently, Zhang et al. [28] proposed two general constructions for PSU protocols with linear computational and communication complexity.

UPSI. Research on UPSI has only begun in the past two years, initially proposed and defined by Badrinarayanan et al. [10]. The UPSI protocol in [10] essentially only supports the addition of elements, while deletion is achieved through a periodic refreshing method, which they refer to as weak deletion. Furthermore, Abadi et al. [29] proposed a delegated UPSI protocol, but this requires the involvement of a third-party cloud server. Recently, Badrinarayanan et al. [11] appeared to address this issue by proposing new UPSI protocols that

support adding and deleting elements. However, their protocols only demonstrate an advantage under very restrictive conditions, i.e., very low bandwidth, tiny update set sizes, and large input sets. Around the same time, Agarwal et al. [13] also proposed a UPSI protocol. However, they did not provide experimental results and code, only theoretical analysis.

3 Preliminaries

3.1 Notation

Let κ denote the computational security parameter, and consider a group $\mathbb{G}$ and a finite field $\mathbb{F}$. We define random oracles $\mathsf{H} : \{0,1\}^* \to \mathbb{G}$ and $\mathsf{F} : \{0,1\}^* \to \mathbb{F}$. The parties $\mathcal{P}_X$ and $\mathcal{P}_Y$ hold the original sets X and Y, respectively. During the i-th execution of the UPSI protocol, the two parties update their sets by adding elements X_i^+ and Y_i^+ and removing elements X_i^- and Y_i^-, respectively. Consequently, the updated sets are expressed as $X_i = (X_{i-1} \setminus X_i^-) \cup X_i^+$ and $Y_i = (Y_{i-1} \setminus Y_i^-) \cup Y_i^+$. We denote the input size and the update size for both parties by N_i and N_i^u, respectively.

3.2 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. [10], who also provided an improved version [11]. In this work, we define the UPSI protocol that supports both addition and deletion operations in Figure 1. In our definition, we require an ideal PSI to perform the initial intersection between both parties. In other words, in each updated intersection computation, the 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 initial private sets X and Y, respectively. Both parties have obtained $I = X \cap Y$ using an ideal $\mathcal{F}^{\mathsf{PSI}}$. Let $X_0 = X$, $Y_0 = Y$, and $I_0 = I$. In the i-th update:

- $\mathcal{P}_X$ updates its set to $X_i = (X_{i-1} \setminus X_i^-) \cup X_i^+$;
- $\mathcal{P}_Y$ updates its set to $Y_i = (Y_{i-1} \setminus Y_i^-) \cup Y_i^+$;
- Both parties receive the updated intersection $I_i = X_i \cap Y_i$ as the output.

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

4 UPSI from Asymmetric PSI and PSU

In this section, we describe our UPSI protocol, which supports arbitrary additions and deletions based on our asymmetric PSI and any efficient PSU protocol.

4.1 Component Description

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

Base PSI. We require a conventional PSI protocol to perform the initial intersection between the two parties. For efficiency in the overall protocol, we recommend using [8] 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, other efficient PSI protocols [5–7] can also be used, which only results in a certain performance loss during the initial intersection computation and does not affect the performance of subsequent updated intersection computations.

Asymmetric PSI. Note that if we directly apply the asymmetric PSI [24], our UPSI will be insecure. For example, if $\mathcal{P}_X$ adds a certain x^+ during one update, deletes it in a subsequent UPSI, and then adds it again, $\mathcal{P}_Y$ would be able to detect this fact because the mask for this x^+ remains the same during the OPRF generation. To avoid such leakage, we should ensure that even for the same x^+, its mask is different in each execution of the UPSI. The same applies to $\mathcal{P}_Y$. Specifically, $\mathcal{P}_X$ and $\mathcal{P}_Y$ should generate the secret values $k_{x,i}$ and $k_{y,i}$ in each execution of UPSI, as this prevents both parties from learning additional information about X^+ and Y^+ across multiple executions of the UPSI. Therefore, we modify the OPRF generation in [24] to the multi-round OPRF generation as shown in Figure 3 to avoid this leakage.

We briefly review the query phase of [24]. Suppose $\mathcal{P}_Y$ is the receiver, $\mathcal{P}_X$ is the sender, and $Y \ll X$. Before proceeding with the query phase, $\mathcal{P}_Y$ and $\mathcal{P}_X$ need to compute $\{\mathsf{F}(k_x, y) \mid y \in Y\}$ and $\{\mathsf{F}(k_x, x) \mid x \in X\}$, respectively, with respect to the OPPF key k_x. Note that in our UPSI, this step is replaced by the protocol in Figure 3. Before proceeding with the query phase, $\mathcal{P}_X$ and $\mathcal{P}_Y$ need to compute $\mathsf{F}(k_x, x)$ and $\mathsf{F}(k_x, y)$, respectively, with respect to the OPPF key k_x. Note that in our UPSI, this step is replaced by the protocol in Figure 3. $\mathcal{P}_Y$ inserts $\{\mathsf{F}(k_x, y)\}$ into a cuckoo hashing B_Y of length N'_Y, while $\mathcal{P}_X$ inserts $\{\mathsf{F}(k_x, x)\}$ into a simple hashing B_X of the same length. $X \cap Y$ can be computed as follows:

$$\{\mathsf{F}(k_x, y)\} \cap \{\mathsf{F}(k_x, x)\} = \bigcup_{i=1}^{N'_Y} B_Y[i] \cap B_X[i].$$

Subsequently, $\mathcal{P}_X$ and $\mathcal{P}_Y$ need to evaluate the polynomial:

$$P(B_Y[i]) = \prod_{x \in B_X[i]} (B_Y[i] - x) \stackrel{?}{=} 0$$

for all $i \in \{1, \cdots, N'_Y\}$ based on homomorphic encryption. Various optimization techniques are also applied, including SIMD packing, partitioning, and windowing. For details, see [24].

PSU. To instantiate our UPSI protocol, we recommend employing the PSU protocol by Zhang et al. [28] due to its linear computational complexity. However, using other PSU protocols [25–27, 30] to instantiate our UPSI protocol is also feasible, as they have linear communication costs and do not result in significant performance loss. We will mention later that instantiating our UPSI using the PSU protocol by Kolesnikov et al. [25] is already very efficient.

4.2 Multi-Round OPRF

We define a multi-round OPRF to prevent information leakage caused by parties using the same elements during the multi-round updates in the UPSI protocol.

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

Without loss of generality, we let $\mathcal{P}_X$ be the sender, and $\mathcal{P}_Y$ be the receiver. In the i-th update:

- $\mathcal{P}_Y$ uses Y_i^+ as input;
- $\mathcal{P}_Y$ receives $\mathsf{F}(k, y_i^+)$ for all $y_i^+ \in Y_i^+$ as the output, and $\mathcal{P}_X$ learns nothing.

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

Note that we impose no restrictions on the Y_i^+ in the multi-round updates. At the same time, $\mathcal{P}_X$ learns no information in each round. In other words, to achieve the ideal functionality shown in Figure 2, even if identical elements exist in Y_i^+ and Y_j^+, $\mathcal{P}_X$ should not be able to observe this fact, where $i \neq j$.

Π^{MROPRF}

$\mathcal{P}_X$ randomly samples k_x. The execution process of the i-th UPSI is as follows.

- $\mathcal{P}_Y$ randomly samples $k_{y,i}$ and computes $Q_{Y_i^+} = \{\mathsf{H}(y_i^+)^{k_{y,i}}\}$ for all $y_i^+ \in Y_i^+$ and sends $Q_{Y_i^+}$ to $\mathcal{P}_X$;
- $\mathcal{P}_X$ computes $E_{Y_i^+} = \{q^{k_x}\}$ for all $q \in Q_{Y_i^+}$ and returns it to $\mathcal{P}_Y$;
- $\mathcal{P}_Y$ computes $F_{Y_i^+} = \{\mathsf{F}(e^{k_{y,i}^{-1}})\}$ for all $e \in E_{Y_i^+}$ as output.

Fig. 3: Our Π^{MROPRF} protocol.

Correctness. The correctness of Π^{MROPRF} is straightforward. It suffices to ensure that, for any two distinct elements y^+ and $y^{+\prime}$, we have $\mathsf{F}(*, y^+) \neq \mathsf{F}(*, y^{+\prime})$. According to [7], we can easily ensure this condition by directly requiring $|\mathbb{F}| \geq 2^\kappa$. In our implementation, we adopt the same setting.

Security. The security of Π^{MROPRF} relies on the DDH assumption.

Theorem 1. *The protocol Π^{MROPRF} realizes $\mathcal{F}_{\mathsf{MROPRF}}$ against a semi-honest adversary.*

Proof. This requires a separate discussion for a corrupted $\mathcal{P}_Y$ and a corrupted $\mathcal{P}_X$.

Corrupted $\mathcal{P}_Y$. $\mathcal{P}_Y$ will receive $E_{Y_i^+}$ from $\mathcal{P}_X$ in the i-th execution. Therefore, the view of $\mathcal{P}_Y$ can be represented as $\mathsf{View}_Y = \{\{Y_i^+\}, \{k_{y,i}\}, \{E_{Y_i^+}\}, \{F_{Y_i^+}\}\}$, where Y_i^+, $k_{y,i}$, and $F_{Y_i^+}$ are the input, random tape, and output of the i-th execution, respectively. Using a sequence of hybrid arguments, we show that the corrupted $\mathcal{P}_Y$ cannot distinguish the elements in $\{E_{Y_i^+}\}$ from random elements in $\mathbb{G}$.

$\mathcal{H}_0$: This is the view of $\mathcal{P}_Y$ in the real execution when it receives $E_{Y_i^+}$.

$\mathcal{H}_{1,j}$: For $j \in \{1, \cdots, N^u\}$, the same as $\mathcal{H}_0$ except that we replace q^{k_x} in $E_{Y_i^+}$ with a random $g_j \in \mathbb{G}$.

$\mathcal{H}_2$: The view of $\mathcal{P}_Y$ as output by the simulator when it finishes receiving $E_{Y_i^+}$.

We argue that $\mathcal{H}_{1,j-1}$ and $\mathcal{H}_{1,j}$ are indistinguishable to $\mathcal{P}_Y$. 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 q, and we let $g^a = g^{k_x}$. $\mathcal{C}$ receives the challenge mask ϵ. Note that $\mathcal{C}$ does not know that ϵ belongs to $\mathcal{H}_{1,j-1}$ or $\mathcal{H}_{1,j}$. $\mathcal{C}$ sends ϵ to $\mathcal{A}$, and then $\mathcal{A}$ determines whether ϵ belongs to $\mathcal{H}_{1,j-1}$ or $\mathcal{H}_{1,j}$. If $c = ab$, the mask $\epsilon = g^c$, otherwise $\epsilon = g_j$ (since g_j is random). If $\mathcal{A}$ judges that ϵ belongs to $\mathcal{H}_{1,j-1}$, 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.

Corrupted $\mathcal{P}_X$. $\mathcal{P}_X$ will receive $Q_{Y_i^+}$ from $\mathcal{P}_Y$ in the i-th execution. Therefore, the view of $\mathcal{P}_X$ can be represented as $\mathsf{View}_X = \{k_x, \{Q_{Y_i^+}\}\}$. The proof is similar to that of $\mathcal{P}_Y$, and we omit the details for brevity.

4.3 Our UPSI

We combine all the components mentioned earlier to construct our UPSI protocol, as illustrated in Figure 4.

Correctness. Proving the correctness of our protocol essentially involves demonstrating that the intersection $I_i = (I_{i-1} \setminus U_i') \cup U_i$ holds for X_i and Y_i, where $U_i = T_i \cup V_i$, $T_i = Y_i^+ \cap X_i$, $V = X_i^+ \cap Y_i$, and $U_i' = (X_i^- \cap I_{i-1}) \cup (Y_i^- \cap I_{i-1})$.

Π^{UPSI}

Initialization:

- $\mathcal{P}_X$ and $\mathcal{P}_Y$ randomly sample k_x and k_y, respectively.
- $\mathcal{P}_X$ and $\mathcal{P}_Y$ locally compute $H_X = \{\mathsf{F}(\mathsf{H}(x)^{k_x})\}$ and $H_Y = \{\mathsf{F}(\mathsf{H}(y)^{k_y})\}$ for all $x \in X$ and $y \in Y$, respectively.
- $\mathcal{P}_X$ and $\mathcal{P}_Y$ invoke an ideal $\mathcal{F}^{\mathsf{PSI}}$, and both parties receive $I = X \cap Y$.

Let $X_0 = X$, $Y_0 = Y$, and $I_0 = I$. The process of the i-th UPSI is as follows.
Deletion:

- If $X_i^- \neq \emptyset$, then $\mathcal{P}_X$ computes $D_{X_i^-} = \{\mathsf{F}(\mathsf{H}(x_i^-)^{k_x})\}$ for all $x_i^- \in X_i^-$, and then updates $H_X = H_X \backslash D_{X_i^-}$.
- If $Y_i^- \neq \emptyset$, then $\mathcal{P}_Y$ computes $D_{Y_i^-} = \{\mathsf{F}(\mathsf{H}(y_i^-)^{k_y})\}$ for all $y_i^- \in Y_i^-$, and then updates $H_Y = H_Y \backslash D_{Y_i^-}$.

Addition:

- If $X_i^+ \neq \emptyset$, then $\mathcal{P}_X$ computes $A_{X_i^+} = \{\mathsf{F}(\mathsf{H}(x_i^+)^{k_x})\}$ for all $x_i^+ \in X_i^+$, and then updates $H_X = H_X \cup A_{X_i^+}$.
- If $Y_i^+ \neq \emptyset$, then $\mathcal{P}_Y$ computes $A_{Y_i^+} = \{\mathsf{F}(\mathsf{H}(y_i^+)^{k_y})\}$ for all $y_i^+ \in Y_i^+$, and then updates $H_Y = H_Y \cup A_{Y_i^+}$.

Compute the intermediate intersection:

- $\mathcal{P}_Y$ invokes the i-th round of Π^{MROPRF} as the receiver with input Y_i^+, and $\mathcal{P}_X$ acts as the sender. Subsequently, $\mathcal{P}_Y$ receives $F_{Y_i^+}$ as output.
- $\mathcal{P}_Y$ and $\mathcal{P}_X$ use the query phase of the asymmetric PSI protocol [24] with $F_{Y_i^+}$ and H_X as inputs. As a result, $\mathcal{P}_Y$ receives $T_i = Y_i^+ \cap X_i$.
- Similarly, $\mathcal{P}_X$ can obtain $V_i = X_i^+ \cap Y_i$.

Compute the intermediate union:

- $\mathcal{P}_X$ and $\mathcal{P}_Y$ invoke an ideal $\mathcal{F}^{\mathsf{PSU}}$, and both parties then receive $U_i = V_i \cup U_i$.
- $\mathcal{P}_X$ and $\mathcal{P}_Y$ locally compute $X_i^- \cap I_{i-1}$ and $Y_i^- \cap I_{i-1}$, respectively.
- $\mathcal{P}_X$ and $\mathcal{P}_Y$ invoke an ideal $\mathcal{F}^{\mathsf{PSU}}$, and both parties then receive $U_i' = (X_i^- \cap I_{i-1}) \cup (Y_i^- \cap I_{i-1})$.

Compute the updated intersection:

- $\mathcal{P}_X$ and $\mathcal{P}_Y$ can locally compute $I_i = (I_{i-1} \backslash U_i') \cup U_i$.

Fig. 4: Our complete UPSI protocol.

We demonstrate this by proving both inclusions:

$$I_i \subseteq (I_{i-1} \setminus U_i') \cup U_i, (I_{i-1} \setminus U_i') \cup U_i \subseteq I_i.$$

$I_i \subseteq (I_{i-1} \setminus U_i') \cup U_i$: For $\forall z \in I_i$, we have $z \in X_i$ and $z \in Y_i$.

Case 1: If $z \in I_{i-1}$, then $z \in X_{i-1}$ and $z \in Y_{i-1}$. Since $z \in X_i$, it must either not be deleted from X_{i-1} (i.e., $z \notin X_i^-$) or be re-added (i.e., $z \in X_i^+$). Similarly, $z \in Y_i$ implies $z \notin Y_i^-$ or $z \in Y_i^+$. Since $z \in I_{i-1}$ and $z \in I_i$, $z \in I_{i-1} \setminus U_i'$ holds if $z \notin U_i'$ (i.e., $z \notin X_i^-$ and $z \notin Y_i^-$).

Case 2: If $z \notin I_{i-1}$, then for $z \in X_i \cap Y_i$ while $z \notin I_{i-1}$, it must be the case that z was added to at least one of the sets: If $z \in Y_i^+$ and $z \in X_i$, then $z \in T_i$. If $z \in X_i^+$ and $z \in Y_i$, then $z \in V_i$. Therefore, $z \in U_i = T_i \cup V_i$.

Combining both cases, we have $z \in (I_{i-1} \setminus U_i') \cup U_i$.

$(I_{i-1} \setminus U_i') \cup U_i \subseteq I_i$: For $\forall z \in (I_{i-1} \setminus U_i') \cup U_i$, we have $z \in I_{i-1} \setminus U_i'$ or $z \in U_i$.

Case 1: If $z \in I_{i-1} \setminus U_i'$, then $z \in I_{i-1}$ and $z \notin U_i'$ hold. Therefore, $z \notin X_i^-$ and $z \notin Y_i^-$ due to $z \notin U_i'$, implying $z \in X_i$ and $z \in Y_i$. Hence, $z \in I_i$.

Case 2: If $z \in U_i$, then $z \in T_i$ or $z \in V_i$ due to $U_i = T_i \cup V_i$. If $z \in T_i = Y_i^+ \cap X_i$, then $z \in Y_i^+$ implies $z \in Y_i$, and $z \in X_i$. If $z \in V_i = X_i^+ \cap Y_i$, then $z \in X_i^+$ implies $z \in X_i$, and $z \in Y_i$.

Combining both subcases, we have $z \in I_i$.

Since both inclusions hold, we conclude that: $I_i = (I_{i-1} \setminus U_i') \cup U_i$.

4.4 Complexity Analysis

We conduct a detailed analysis of the complexities at each phase of the UPSI protocol to determine its overall complexity. 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 [5,6,8]. The repeated execution of the remaining phases alone constitutes our UPSI protocol. For the sake of discussion, we assume $|X| = |Y| = N$ and $|X_i^+| = |Y_i^+| = |X_i^-| = |Y_i^-| = N^u$.

Deletion. The complexity of this phase is $\mathcal{O}(N^u + N^u \log N)$. 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. This phase consists of two invocations of Π^{MROPRF} and two query phases of the asymmetric PSI [24]. Clearly, both the computational and communication complexities of Π^{MROPRF} are $\mathcal{O}(N^u)$. Moreover, the computational and communication complexities of the query phase are $\mathcal{O}(\sqrt{N})$ and $\mathcal{O}(\log^2 N)$, respectively. Therefore, the overall computational and communication complexities for this entire phase are $\mathcal{O}(N^u + \sqrt{N})$ and $\mathcal{O}(N^u + \log^2 N)$, respectively.

Compute the intermediate union. This phase mainly involves two invocations of the PSU protocol. Note 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 [25–28, 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 can be summarized as $\mathcal{O}(N^u \cdot \log N + \sqrt{N})$ and $\mathcal{O}(N^u + \log^2 N)$, respectively.

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 [10, 11, 13].

Theorem 2. *The protocol* Π^{UPSI} *realizes* $\mathcal{F}_{\mathsf{UPSI}}$ *against a semi-honest adversary.*

Proof. Our UPSI protocol Π^{UPSI} is obviously secure if there exist ideal functionalities $\mathcal{F}_{\mathsf{PSI}}$, $\mathcal{F}_{\mathsf{PSU}}$, and $\mathcal{F}_{\mathsf{MROPRF}}$, while the messages exchanged in the query phase of [24] are encrypted ciphertexts of homomorphic encryption.

5 Evaluation

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.

5.1 Experimental Setup

We used two Docker 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[1], which is a C++ library that contains common cryptography, network and I/O modules. The computational security parameter is set to $\kappa = 128$, and performance is evaluated under LAN and WAN settings. The LAN connection is simulated with a 0.2 ms RTT network latency and 1 Gbps network bandwidth. For the WAN connection, the Linux "tc" command is used, and the bandwidth is configured following the setup of Badrinarayanan et al. [11], with results presented at 200 Mbps, 50 Mbps, and 5 Mbps, along with an RTT latency of 80 ms. The group $\mathbb{G}$ is instantiated using the Four$\mathbb{Q}$ curve [31], and SHA-512 is employed for H, while BLAKE3 is used for F. All protocols in our experiments are evaluated using 128-bit elements. Regarding asymmetric PSI,

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

we use the APSI library[2], which is the official implementation of [24]. We use the PSU protocol proposed by Kolesnikov et al. [25] to instantiate our UPSI instead of the linear-computation-cost PSU [28], 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 [25] based on the experimental results. Note that if the PSU protocol by Zhang et al. [28] 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. [8]. For the sake of presenting the results, we assume $|X| = |Y| = N$ and $|X_i^+| = |Y_i^+| = |X_i^-| = |Y_i^-| = N^u$.

5.2 The Performance of Our UPSI

First, we present the costs of our UPSI protocol during the initialization phase, which is a pre-computation phase that can only be executed once. The costs of this phase depend only on N and are linear. This phase primarily requires a single execution of the base PSI protocol and pre-computation by both parties for the masks used in the asymmetric PSI.

We present the costs of the initialization phase of our UPSI under different values of N and various network environments in Table 1. Since the initialization phase requires executing the base PSI, it can become relatively slow when the bandwidth is low due to the significant communication overhead it incurs. For example, when $N = 2^{22}$ and the bandwidth is 5 Mbps, the initialization phase takes 417.57 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 phase. We also do not include the costs of the initialization phase in the subsequent UPSI evaluations. Note that the communication overhead for the base PSI is slightly higher than executing a one-way PSI protocol by Raghuraman et al. [8], as the party receiving the intersection needs to send it to the other party.

Table 1: Communication cost (in MB) and running time (in seconds) of the initialization phase under different values of N and various network bandwidths.

N	Comm. (MB)	Total Running Time (s)			
		LAN	200Mbps	50Mbps	5Mbps
2^{18}	11.19	6.74	7.7	9.38	29.53
2^{20}	43.89	19.6	22.19	28.77	107.77
2^{22}	174.78	67.62	76.75	102.97	417.57

[2] https://github.com/microsoft/APSI
[3] https://github.com/alibaba-edu/mpc4j

Next, in Table 2, we present the costs of our protocol under different values of N and N^u and various network environments. With a bandwidth of 5 Mbps, both parties can complete the UPSI protocol in 4.05 seconds when $N = 2^{20}$ and $N^u = 2^{10}$. When dealing with large-scale data, our protocol can also quickly complete the computation of the updated intersection. For example, when $N = 2^{22}$ and $N^u = 2^{10}$, our UPSI protocol completes the intersection computation in 9.14, 9.81, 10.6, and 20.09 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 $N = 2^{22}$ and $N^u = 2^{13}$, our protocol requires only 77.09, 78.92, 83.2, and 134.6 seconds under LAN, 500 Mbps, 50 Mbps, and 5 Mbps WAN settings, respectively.

Table 2: 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^{18}	2^9	2.72	1.99	2.53	2.93	7.84
	2^{10}	4.25	4.51	5.12	5.76	13.4
	2^{11}	5.34	9.15	9.82	10.62	20.22
	2^{12}	8.27	14.69	15.5	16.74	31.62
2^{20}	2^{10}	5.21	5.22	5.88	6.66	16.04
	2^{11}	9.07	7.79	8.64	10.0	26.33
	2^{12}	12.58	16.35	17.38	19.27	41.9
	2^{13}	20.71	27.76	29.2	32.3	69.58
2^{22}	2^{10}	5.27	9.14	9.81	10.6	20.09
	2^{11}	8.09	14.66	15.46	16.68	31.23
	2^{12}	13.38	37.41	38.48	40.48	64.56
	2^{13}	28.55	77.09	78.92	83.2	134.6

5.3 Comparison with State-of-The-Art Protocols

We compare our UPSI protocol with the state-of-the-art conventional PSI protocol [8] and UPSI protocol [11]. We present the communication cost and running time of our protocol in comparison with [8, 11] in Table 3.

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

Communication Improvement. Our protocol outperforms RR22 [8] by $8 \sim 42\times$ in all settings. When $N = 2^{22}$ and $N^u = 2^{10}$, the communication cost of RR22 is 206.65 MB, whereas our protocol requires only 5.27 MB. Furthermore, our protocol achieves a $2 \sim 6\times$ reduction compared to the version that supports only addition in [11]. When $N = 2^{20}$ and $N^u = 2^{10}$, it requires 29.61 MB of

Table 3: 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 [8]	43.88	1.7	4.3	10.88	89.87
	2^8	BMS$^+$24 [11] (addition-only)	7.57	22.8	23.58	24.71	38.34
	2^9		14.87	43.9	45.04	47.27	74.04
	2^{10}		29.61	87.6	89.48	93.92	147.22
	2^8	BMS$^+$24 [11] (addition & deletion)	922	1398.4	1445.25	1584.6	3256.8
	2^9		1845	2696	2788.65	3065.4	6386.4
	2^{10}		3687	5543.5	5727.75	6280.8	-
	2^8	Ours	4.21	3.13	3.74	4.37	11.95
	2^9		4.86	4.95	5.59	6.32	15.07
	2^{10}		5.21	5.22	5.88	6.66	16.04
2^{22}	-	RR22 [8]	206.65	8.17	18.87	49.77	420.57
	2^8	BMS$^+$24 [11] (addition-only)	8.03	23.6	24.4	25.61	40.06
	2^9		15.97	47.2	48.4	50.79	79.54
	2^{10}		31.5	92.4	94.38	99.1	155.8
	2^8	BMS$^+$24 [11] (addition & deletion)	927	1603.7	1650.45	1789.5	3458.1
	2^9		1855	3305	3398.15	3676.4	-
	2^{10}		3711	6874	7059	-	-
	2^8	Ours	4.89	7.11	7.76	8.49	17.29
	2^9		5.01	7.8	8.45	9.2	18.22
	2^{10}		5.27	9.14	9.81	10.6	20.09

communication, whereas our UPSI needs only 5.27 MB. Our protocol reduces communication overhead by 190 $\sim$ 707$\times$ compared to [11] 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 5.21 MB.

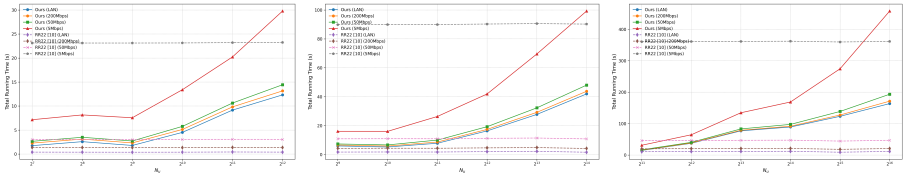
Computation Improvement. Our protocol achieves up to a 16$\times$ reduction in computation overhead compared to the version that supports only addition in [11]. Specifically, when $N = 2^{20}$ and $N^u = 2^{10}$, it takes 87.6 seconds to complete the protocol, whereas our UPSI protocol requires only 5.22 seconds in the LAN setting. Furthermore, our protocol reduces computation overhead by 255 $\sim$ 1061$\times$ compared to [11] that supports both addition and deletion. For example, when $N = 2^{22}$ and $N^u = 2^{10}$, it takes 6874 seconds to complete the protocol, whereas our UPSI protocol still requires only 9.14 seconds.

End-to-End. Our protocol can be 3 $\sim$ 9$\times$ faster than the version that supports only addition in [11]. For example, when $N = 2^{20}$ and $N^u = 2^{10}$, the

latter requires 89.48, 93.92, and 147.22 seconds to complete the protocol under 200 Mbps, 50 Mbps, and 5 Mbps, respectively, whereas our protocol only needs 5.88, 6.66, and 16.04 seconds under the same bandwidths. Our protocol can be up to 974× faster than [11] 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 5.88 seconds with a bandwidth of 200 Mbps. Furthermore, our UPSI also has an advantage in overall running time compared to [8] in most settings. When the bandwidth is 5 Mbps, our protocol can be $5 \sim 24\times$ faster than [8]. Specifically, when $N = 2^{22}$ and $N^u = 2^{10}$, [8] takes 420.57 seconds to complete the intersection computation, whereas our UPSI requires only 17.29 seconds.

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 [10,11,13] 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 conventional two-party PSI protocol [8].



(a) $N = 2^{18}$ (b) $N = 2^{20}$ (c) $N = 2^{22}$

Fig. 5: Comparison with the state-of-the-art conventional PSI [8] under different bandwidths and updated set sizes when $N \in \{2^{18}, 2^{20}, 2^{22}\}$.

As shown in Figure 5, we present a comparison of our UPSI protocol with [8] under different bandwidths and updated set sizes when $N \in \{2^{18}, 2^{20}, 2^{22}\}$. From Figures 5a to 5c, we can see that the performance of our protocol under different N shows a consistent increasing trend as N^u increases. Let N_t^u be a threshold for N^u, beyond which it becomes more efficient to re-run [8] rather than executing our UPSI protocol. We can determine the different values of N_t^u under various bandwidths by observing when the running time of our UPSI exceeds that of [8] under the same bandwidth. After knowing the value of N_t^u, if both parties need to update a set larger than N_t^u, 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 [8]. 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 [11] that supports addition and deletion has an actual N_t^u 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 setting of 200 Mbps, the N_t^u for this configuration is 2^{11}, while [11] is only efficient compared to [8] under very low bandwidth and small update set sizes. Moreover, we can observe that as N increases, N_t^u also tends to increase in a certain pattern. For example, when N is fixed, the value of N_t^u 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.

6 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.

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. 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.

6. 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.
7. 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.
8. 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.
9. 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.
10. Saikrishna Badrinarayanan, Peihan Miao, and Tiancheng Xie. Updatable private set intersection. *Proc. Priv. Enhancing Technol.*, 2022(2):378–406, 2022.
11. 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.
12. 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.
13. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.

19. 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.
20. 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.
21. 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.
22. 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.
23. 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.
24. 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.
25. 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.
26. 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.
27. 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.
28. 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.

29. 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.
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. Craig Costello and Patrick Longa. Four$\mathbb{Q}$: Four-dimensional decompositions on a $\mathbb{Q}$-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.
32. Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, et al. Private matching for compute. *IACR Cryptol. ePrint Arch.*, page 599, 2020.
33. 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.
34. 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.
35. 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.
36. 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.
37. 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.

A Applications

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

Vertical Federated Learning. Vertical federated learning [9] 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 [32]: "*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 2, 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 [33, 34]. 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 [35–37]. 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.