Updatable Private Set Union: Generic Construction with Efficient Instantiation

Seongbong Choi and Hyung Tae Lee(✉)

Chung-Ang University, Republic of Korea
{welq2st,hyungtaelee}@cau.ac.kr

Abstract. Private set union (PSU) allows two parties to compute the union of their private sets without revealing their intersection. In many real-world applications, parties' datasets undergo frequent updates as elements are added or removed over time. Existing PSU protocols, however, must recompute the entire union from scratch whenever either party's set changes. This becomes highly inefficient when updates are small or frequent relative to the original set sizes. In this paper, we introduce the first updatable PSU (uPSU) protocol for the standard two-party setting, which supports efficient incremental updates. We present a systematic classification of all possible update scenarios, which shows that only a small subset of updated elements actually modify the union, and establish the leakage baseline for uPSU. Based on these classification and leakage baseline, we provide a generic construction for uPSU that uses existing PSI and a tagged variant of PSU as building blocks. We prove security against semi-honest adversaries in the simulation-based model, and guarantee that incremental updates reveal no more information than a fresh execution of a standard PSU protocol on the updated sets. We instantiate and implement our generic construction using Kim et al.'s PSU protocol (ACM SAC 2026) and Raghuraman and Rindal's PSI protocol (ACM CCS 2022), demonstrating significant performance improvements over full recomputation of the union, even though its cost still depends on the original set size rather than purely on the update size. For set size $n = 2^{20}$ and update size $t = 2^{12}$, our protocol achieves a 14.1–45.4× speedup with a 4.8–59.1× communication reduction over full recomputation using baseline PSU protocols.

Keywords: Private set union · updatable private set union · generic construction · instantiation

1 Introduction

Private set union (PSU) allows two parties to compute the union of their private sets without revealing their intersection. Unlike private set intersection (PSI), which reveals shared elements, PSU enables parties to aggregate data while hiding which elements belong to both sets. This functionality is crucial in scenarios where the intersection itself is sensitive information that could leak strategic or proprietary knowledge.

PSU has numerous applications in security and privacy-sensitive domains. These include cyber risk assessment [23, 16, 29, 11], private identifier generation [7, 14], network event aggregation [8], and privacy-preserving data integration [21, 22]. In all these scenarios, revealing the intersection would expose sensitive operational intelligence or strategic capabilities.

In many real-world applications, parties frequently update their datasets as elements are added or removed over time. However, existing PSU protocols assume static datasets and must recompute the entire union from scratch whenever either party's set changes. Recomputing the entire union becomes highly inefficient when updates are small or frequent relative to the original set sizes. This motivates the need for protocols that can handle incremental updates efficiently.

1.1 Problem Statements

Recent advances have significantly improved the efficiency of PSU, with state-of-the-art protocols [34, 33, 18, 30, 19, 26, 27] achieving linear communication complexity for large datasets. Despite this progress, all existing PSU protocols [21, 13, 6, 22, 17, 31, 34, 10, 33, 9, 30, 19, 26, 27] are limited to static datasets. In other words, when either party modifies its set—even by a small number of insertions or deletions—both parties must rerun the entire protocol from scratch. Consequently, no existing protocol can reuse previously computed unions or intermediate states, which leads to substantial inefficiencies in dynamic environments where updates are frequent but small relative to the size of the datasets.

Motivations. Consider collaborative scenarios in which multiple parties maintain evolving datasets that must be periodically aggregated while preserving privacy. In cyber risk assessment, security organizations maintain IP blacklists that evolve continuously as new threats are detected and old entries are removed [16, 29]. For private identifier generation, organizations assign identifiers to the union of their customer bases to enable coordinated operations, requiring recomputation of the union as customer databases evolve [7, 14]. In network event aggregation, monitoring systems continuously collect security events and network statistics that must be aggregated in real-time without revealing sensitive network configurations [8]. Similar update patterns arise across other scenarios, including privacy-preserving data integration [21, 22].

These scenarios share common characteristics. According to real-world studies [16, 29], organizations typically maintain datasets containing thousands to hundreds of thousands of entries, with updates occurring frequently throughout the day. These updates often involve adding and removing hundreds to thousands of entries, which constitute only a small fraction of the total dataset size. Running a full PSU protocol for each update requires reprocessing the entire dataset repeatedly, even when changes are incremental. The computational overhead becomes prohibitive, forcing parties to either update less frequently, accept stale data, or abandon privacy-preserving collaboration entirely. Yet no existing PSU protocol addresses this limitation.

Limitations of Existing Protocols. Existing PSU protocols restart the entire computation from scratch upon each update, incurring the same cost regardless

of the update size. For instance, when updates constitute only 1% of the total dataset, existing protocols impose a 100× overhead relative to the update size. To the best of our knowledge, only one prior work addresses updates in the PSU domain. Chung et al. [11] proposed a PSU protocol that supports updates, but it operates in a delegated multi-party setting where multiple clients outsource computation to the cloud. This fundamentally different architectural model has distinct security assumptions and deployment requirements, which precludes direct comparison with standard two-party PSU protocols.

Challenges of Designing Updatable PSU. Since PSI and PSU are closely related functionalities, one might expect that updatable PSU (uPSU) can be constructed by direct application of techniques developed for updatable PSI (uPSI). However, the two functionalities differ fundamentally in the nature of their output leakage, which introduces unique challenges in the updatable setting.

In uPSI, the receiver learns only elements within its own set, so updates that do not affect the intersection are automatically hidden. This containment property makes it relatively straightforward to reason about the information revealed by each update.

In uPSU, however, the receiver learns elements outside its own set, i.e., $X \setminus Y$, from the union $X \cup Y$, where X and Y denote the sender's and receiver's sets, respectively. As a result, every update to either party's set can potentially alter the receiver's view in ways that depend intricately on both parties' current states. A key challenge is that different combinations of updates may result in the same union, making it non-trivial to determine what information is inevitably revealed and what must be hidden. For instance, when the sender adds an element and the receiver simultaneously deletes it, a naïve protocol may reveal not just the union change, but also the timing and origin of the sender's operations, revealing more than what would be learned by recomputing the union from scratch. Addressing this requires a precise characterization of which update scenarios actually modify the union and what leakage is inherent to each case, which is a more fundamental challenge than it may initially appear.

1.2 Our Solutions

We present the first uPSU protocol for the standard two-party setting. The central challenge is not only to identify which updates modify the union, but also to ensure that incremental updates reveal no more than what is already implied by naïve PSU recomputation under the receiver's accumulated view. We address this through a systematic analysis of all possible update scenarios and establish the leakage baseline.

Structure of the update. We classify each updated element by its membership in both parties' sets, yielding 12 disjoint cases that exhaustively cover all possible update interactions. Only 6 of these cases modify the union; the remaining 6 leave it unchanged and are inherited from the previous union without any action. The 6 changing cases further split into two groups by membership condition. The sender's additions and deletions share a single condition and can be handled

jointly on the sender's side. The remaining cases involve intersections with the receiver's own updates and must therefore be resolved on the receiver's side, where they pair naturally into additions and deletions. This analysis determines not only which elements must be processed, but also the precise leakage that any uPSU protocol must exhibit.

A natural fit with existing primitives. The sender's side maps directly onto the core step of standard PSU, in which the sender obliviously transmits elements absent from the receiver's set. Attaching a tag bit to each transmitted element to distinguish additions from deletions yields a *tagged PSU* without any structural change to the underlying protocol. The receiver's side uses state from the previous PSU execution for additions and a single PSI invocation for deletions. This design also matches the leakage baseline: the sender's unique additions and deletions are revealed only when unavoidable, while the remaining union-changing cases are exposed only in aggregated form, without separately revealing information. Once the leakage baseline is established, our construction builds entirely on PSI and a tagged variant of PSU as cryptographic building blocks, without requiring any new primitives, and directly inherits the efficiency of state-of-the-art constructions of these building blocks.

1.3 Our Contributions

Our contributions are summarized as follows:

- We provide the first systematic classification of update scenarios in uPSU into 12 disjoint cases based on each element's membership in both parties' original sets, additions, and deletions in Section 3. Only six of these twelve cases actually modify the union, and this classification directly enables the design of our generic construction by identifying exactly which elements require computation. We also establish the leakage baseline for uPSU by analyzing the information that the receiver learns from naïve PSU recomputation.
- Based on the classification and leakage baseline, we present a generic uPSU protocol that uses existing PSI and a tagged variant of PSU as building blocks in Section 4.
- We prove that our protocol securely realizes the uPSU functionality $\mathcal{F}_{\mathsf{uPSU}}$ in the $(\mathcal{F}_{\mathsf{PSU}}, \mathcal{F}_{\mathsf{PSI}})$-hybrid model against semi-honest adversaries. Specifically, we show that our protocol achieves the same security guarantees as recomputing the union from scratch using a PSU protocol on the updated sets X_{new} and Y_{new}.
- We provide a concrete instantiation of our generic construction using a tagged variant of Kim et al.'s PSU protocol [19] and Raghuraman and Rindal's PSI protocol [28], and implement it in C++. We then compare our protocol against three full-recomputation PSU baselines: JSZG24 [18], TBZ+25 [30], and KLS26 [19]. For set size $n = 2^{20}$ and $t = 2^{12}$, where each party performs t additions and t deletions in the LAN setting, it achieves

a 14–45× speedup with a 4.8–59× reduction in communication, and it retains a 2.2–6.9× speedup even when the update ratio $2t/n$ reaches 25% at $t = 2^{17}$, as detailed in Section 5. Because both phases are still configured for the original set size n, the cost scales with n rather than purely with t, which we leave as an open problem in Section 6.

1.4 Related Work

This section reviews prior work on PSU and the more closely related line of update-supporting private set operation protocols.

Private Set Union. Since Kissner and Song proposed the first PSU protocol [21], various PSU constructions have been proposed. Early works [21, 13, 6, 12] relied on computationally expensive operations such as additively homomorphic encryption or generic MPC, resulting in limited scalability. Later, Kolesnikov et al. [22] introduced the reverse private membership test (RPMT) using symmetric-key techniques and provided the first scalable PSU protocol based on RPMT. Following this breakthrough, several improvements were made in different directions. Garimella et al. [14] proposed a framework for private set operations including PSU based on oblivious switching networks. Subsequently, Jia et al. [17] identified security issues in the split-execute-assemble paradigm of [22] and proposed a shuffle-based approach. Gordon et al. [15] proposed a two-party PSU whose receiver message can be reused across multiple executions. Unlike the incremental updates we target, such reusability amortizes a fixed receiver message across repeated runs while still recomputing the union over the full sets. Zhang et al. [34] formalized the multi-query RPMT functionality and achieved PSU with linear complexity in both communication and computation. Chen et al. [10] further improved PSU efficiency. Tu et al. [31] addressed the unbalanced setting using fully homomorphic encryption.

However, Jia et al. [18] claimed that RPMT-based PSU protocols suffer from during-execution leakage (DEL), where a receiver can abort mid-execution to learn partial intersection information, and proposed a scalable countermeasure. Building on this, Tu et al. [30] proposed enhanced PSU protocols preventing DEL in both balanced and unbalanced scenarios through permuted non-membership conditional randomness generation. Chandran et al. [9] proposed a PSU protocol based on circuit-PSI with a combine-and-permute approach. Concurrently, Kim et al. [19] proposed PSO protocols based on circuit-PSI that maintain membership bits in secret-shared form throughout protocol execution to prevent DEL, and noted that Chandran et al.'s protocol [9] remains susceptible to DEL due to its RPMT-like two-step structure.

Piske and Trieu [26] proposed the first PSU protocol based on invertible Bloom lookup tables, departing from the RPMT paradigm and relying only on oblivious transfers and oblivious pseudorandom functions, and demonstrated that PSU can be achieved with runtimes comparable to those of state-of-the-art PSI protocols. Pu et al. [27] revisited the PSU functionality and argued that DEL is not a well-founded notion in the semi-honest model, and presented the

first practical PSU protocols secure against malicious adversaries with overhead comparable to semi-honest protocols.

Private Set Operation Supporting Updates. Early work by Kiss et al. [20] explored PSI that can be updated in mobile settings. However, their work lacks formal security analysis of the leakage when datasets are updated. Abadi et al. [1] proposed a multi-party delegated solution for PSI supporting updates. Badrinarayanan et al. [5] first formalized uPSI with concrete security definitions, and proposed solutions for insertions and weak deletions. In a follow-up work, Badrinarayanan et al. [4] provided solutions for arbitrary deletions and extended functionalities using oblivious data structures. Wang et al. [32], Agarwal et al. [2], and Ling et al. [24] proposed various uPSI constructions with different trade-offs. Recently, Alborch et al. [3] identified privacy leakages in existing uPSI constructions and proposed a generic framework based on circuit-PSI to mitigate them. Liu et al. [25] proposed the first symmetric-key uPSI protocol by extending the PSI paradigm based on vector oblivious linear evaluation (VOLE) to the updatable setting, achieving orders-of-magnitude improvements in computational efficiency over prior uPSI work.

Meanwhile, Chung et al. [11] extended Abadi et al.'s multiparty updatable delegated PSI protocol [1] to the PSU setting, where multiple clients outsource computation to a cloud server. To the best of our knowledge, this is the only prior PSU work that supports updates.

2 Preliminaries

In this section, we introduce the notations, security definitions, and building blocks used throughout the paper.

2.1 Notations

Throughout this paper, we denote the computational security parameter by κ and the statistical security parameter by λ. For a positive integer n, $[n]$ represents the set $\{1, 2, \ldots, n\}$. $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$ denotes that $\mathcal{D}_1$ and $\mathcal{D}_2$ are computationally indistinguishable. For a set X_{old}, we define X_{add} and X_{del} as the sets of elements to be added to and deleted from X_{old}, respectively, where $X_{\mathsf{del}} \subseteq X_{\mathsf{old}}$ and $X_{\mathsf{add}} \cap X_{\mathsf{old}} = \emptyset$. We write the retained set as $X_{\mathsf{ret}} = X_{\mathsf{old}} \setminus X_{\mathsf{del}}$ and the updated set as $X_{\mathsf{new}} = X_{\mathsf{add}} \cup X_{\mathsf{ret}}$.

2.2 Definitions

Throughout the paper, we consider the semi-honest model. We next introduce the security definitions and cryptographic primitives used in our construction.

Definition 1 (Semi-Honest Security). *Let Π be a protocol between parties $\mathcal{S}$ with input X and $\mathcal{R}$ with input Y that computes a functionality $f(X, Y) = (f_{\mathcal{S}}(X, Y), f_{\mathcal{R}}(X, Y))$. Let $\mathsf{View}^{\Pi}_{\mathcal{P}}(X, Y)$ denote the view of party $\mathcal{P} \in \{\mathcal{S}, \mathcal{R}\}$*

during the execution of Π. We say that Π is secure against semi-honest adversaries if there exist probabilistic polynomial-time (PPT) simulators $\mathsf{Sim}_{\mathcal{S}}$ and $\mathsf{Sim}_{\mathcal{R}}$ that satisfy the following:

$$\{\mathsf{Sim}_{\mathcal{S}}(1^{\kappa}, X, f_{\mathcal{S}}(X,Y))\} \overset{c}{\approx} \{\mathsf{View}^{\Pi}_{\mathcal{S}}(X,Y)\} \text{ and}$$
$$\{\mathsf{Sim}_{\mathcal{R}}(1^{\kappa}, Y, f_{\mathcal{R}}(X,Y))\} \overset{c}{\approx} \{\mathsf{View}^{\Pi}_{\mathcal{R}}(X,Y)\}.$$

We now recall the ideal functionalities for PSI and PSU that serve as building blocks in our protocol.

Definition 2 (Ideal Functionality $\mathcal{F}_{\mathsf{PSI}}$ of PSI). *Let X be the sender $\mathcal{S}$'s input set of size n, and let Y be the receiver $\mathcal{R}$'s input set of size m. $\mathcal{R}$ receives the intersection $I = X \cap Y$.*

Definition 3 (Ideal Functionality $\mathcal{F}_{\mathsf{PSU}}$ of PSU). *Let X be the sender $\mathcal{S}$'s input set of size n, and let Y be the receiver $\mathcal{R}$'s input set of size m. $\mathcal{R}$ receives the union $U = X \cup Y$.*

Most existing PSU protocols share a common structure [22, 14, 17, 34, 10, 19]: for each sender element, the two parties privately test whether it belongs to Y, and the protocol obliviously transmits only the elements absent from Y, namely $X \setminus Y$, to the receiver. As a result, the receiver learns $X \setminus Y$ but not the intersection $X \cap Y$. Building on this structure, we employ a *tagged PSU* that attaches a tag to each element to indicate its intended role: the sender tags each element of X before transmission, and the tag is carried along to the transmitted elements $X \setminus Y$ without any change to how the protocol selects them. Hence, tagged PSU is supported by any existing PSU with minimal modification. In our uPSU, we use two tags $+$ and $-$ to represent additions and deletions, respectively. Additional tags can be used analogously to represent other roles if needed.

Definition 4 (Ideal Functionality $\mathcal{F}_{\mathsf{tPSU}}$ of tagged PSU). *Let X_1 and X_2 be the sender $\mathcal{S}$'s input sets of sizes n_1 and n_2, tagged with $\mathtt{t}_1$ and $\mathtt{t}_2$, respectively, and let Y be the receiver $\mathcal{R}$'s input set of size m. $\mathcal{R}$ receives $(X_1 \setminus Y, \mathtt{t}_1)$ and $(X_2 \setminus Y, \mathtt{t}_2)$.*

3 Systematic Classification for uPSU

In this section, we classify all updated elements to determine which elements modify the union. We also analyze the information that the receiver learns from naïve PSU recomputation, which establishes the privacy baseline for uPSU.

3.1 Case Classification of Update Cases

When a party updates its set, the change in membership of each element is classified into four cases: *addition*, where an element is added to the set for the first time; *retention*, where an element remains in the set; *deletion*, where an

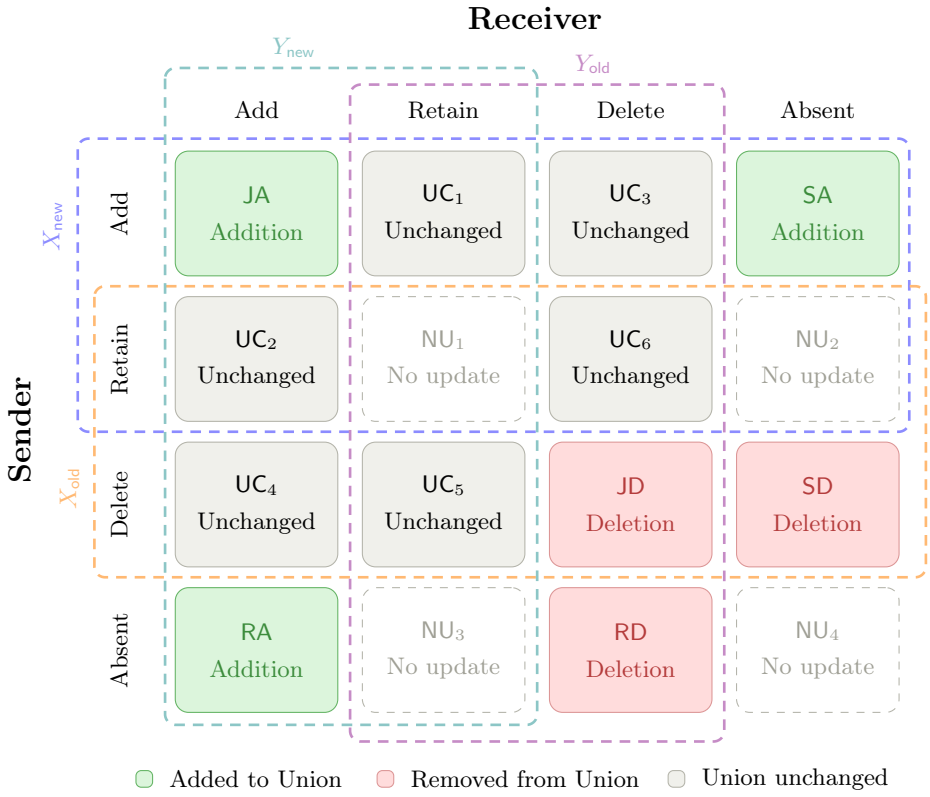

Fig. 1: Union update result based on sender and receiver actions. JA: joint addition, SA: sender's unique addition, RA: receiver's unique addition, JD: joint deletion, SD: sender's unique deletion, RD: receiver's unique deletion, UC: unchanged, NU: no update.

element is removed from the set; and *absence*, where an element is not present before or after the update.

Let $X_{\mathsf{add}}, X_{\mathsf{del}}$ denote the sender's addition and deletion sets, and let $Y_{\mathsf{add}}, Y_{\mathsf{del}}$ denote the receiver's addition and deletion sets. After updates, the new sets are $X_{\mathsf{new}} = X_{\mathsf{add}} \cup X_{\mathsf{ret}}$ and $Y_{\mathsf{new}} = Y_{\mathsf{add}} \cup Y_{\mathsf{ret}}$, where $X_{\mathsf{ret}} = X_{\mathsf{old}} \setminus X_{\mathsf{del}}$ and $Y_{\mathsf{ret}} = Y_{\mathsf{old}} \setminus Y_{\mathsf{del}}$, yielding $U_{\mathsf{new}} = X_{\mathsf{new}} \cup Y_{\mathsf{new}}$. Whether an updated element affects the union depends entirely on its membership in the other party's sets. By cross-tabulating the four cases of the sender and receiver, we obtain a 4×4 matrix of combinations, as illustrated in Figure 1. Excluding the four cases $\mathsf{NU}_1, \ldots, \mathsf{NU}_4$ (no-update cases) in which neither party performs any update, this yields 12 disjoint cases that exhaustively cover all possible update interactions. Among them, only six cases modify the union: JA (joint addition), SA (sender's unique addition), and RA (receiver's unique addition) introduce new elements to the union, while JD (joint deletion), SD (sender's unique deletion), and RD (receiver's unique deletion) remove elements from the union. The

remaining six cases $\mathsf{UC}_1, \ldots, \mathsf{UC}_6$ (unchanged cases) leave the union unchanged, as each element remains in U_{new} through at least one party. Thus, the updated union is

$$U_{\mathsf{new}} = U_{\mathsf{add}} \cup (U_{\mathsf{old}} \setminus U_{\mathsf{del}}), \tag{1}$$

where $U_{\mathsf{add}} = \mathsf{JA} \cup \mathsf{SA} \cup \mathsf{RA}$ and $U_{\mathsf{del}} = \mathsf{JD} \cup \mathsf{SD} \cup \mathsf{RD}$. Therefore, computing the updated union reduces to obtain $U_{\mathsf{add}} = \mathsf{JA} \cup \mathsf{SA} \cup \mathsf{RA}$ and $U_{\mathsf{del}} = \mathsf{JD} \cup \mathsf{SD} \cup \mathsf{RD}$.

A closer examination of Figure 1 reveals that the membership structure alone determines how each union-changing case should be grouped and which reference set governs it. The following design insights (DI) make this structure explicit.

- (DI 1: *A common sender-side condition for* SA *and* SD.) The sender's additions and deletions that actually change the union, namely SA and SD, satisfy exactly the same membership condition: non-membership in $Y_{\mathsf{old}} \cup Y_{\mathsf{add}}$. The sender can therefore handle both cases in a single computation, with their effect distinguished only at the output.
- (DI 2: *A receiver-side grouping of* JA*,* JD*,* RA*, and* RD.) The cases $\mathsf{JA} = X_{\mathsf{add}} \cap Y_{\mathsf{add}}$ and $\mathsf{JD} = X_{\mathsf{del}} \cap Y_{\mathsf{del}}$ correspond to intersections of both parties' updates, which PSU must inherently keep hidden. Any attempt to handle them on the sender's side would expose intersection information. On the receiver's side, however, they are naturally absorbed into RA and RD without any such distinction. In particular, $\mathsf{JA} \cup \mathsf{RA} = Y_{\mathsf{add}} \setminus X_{\mathsf{old}}$ and $\mathsf{JD} \cup \mathsf{RD} = Y_{\mathsf{del}} \setminus X_{\mathsf{new}}$. Thus, the receiver resolves these four cases through two simple set operations, while JA and JD remain hidden.

Together, (DI 1) and (DI 2) reduce the entire update to four set differences: $\mathsf{SA} = X_{\mathsf{add}} \setminus (Y_{\mathsf{old}} \cup Y_{\mathsf{add}})$ and $\mathsf{SD} = X_{\mathsf{del}} \setminus (Y_{\mathsf{old}} \cup Y_{\mathsf{add}})$ on the sender's side, and $\mathsf{JA} \cup \mathsf{RA} = Y_{\mathsf{add}} \setminus X_{\mathsf{old}}$ and $\mathsf{JD} \cup \mathsf{RD} = Y_{\mathsf{del}} \setminus X_{\mathsf{new}}$ on the receiver's side.

3.2 Leakage Baseline of uPSU

The leakage analysis of uPSU should account for the receiver's accumulated knowledge across invocations. Specifically, U_{old} retained from the previous PSU execution contains $X_{\mathsf{old}} \setminus Y_{\mathsf{old}}$, so every updated element should be analyzed with respect to its membership in both the current inputs and this inherited information. By comparing the two outputs U_{old} and U_{new} together with the receiver's updated input sets $(Y_{\mathsf{old}}, Y_{\mathsf{add}}, Y_{\mathsf{del}})$, the receiver learns different levels of information about the sender's operations. We formulate the resulting leakage as security properties (SPs) by focusing on the columns corresponding to the receiver's updated sets, as summarized in Figure 2.

- (SP 1: *Column 1*) JA, UC_2, UC_4, and RA are all contained in U_{new}. From U_{old} and U_{new}, the receiver can distinguish $\mathsf{UC}_2 \cup \mathsf{UC}_4$ from $\mathsf{JA} \cup \mathsf{RA}$. However, it should remain indistinguishable whether an element in $\mathsf{UC}_2 \cup \mathsf{UC}_4$ belongs to UC_2 or UC_4. Likewise, it should remain indistinguishable whether an element in $\mathsf{JA} \cup \mathsf{RA}$ belongs to JA or RA.

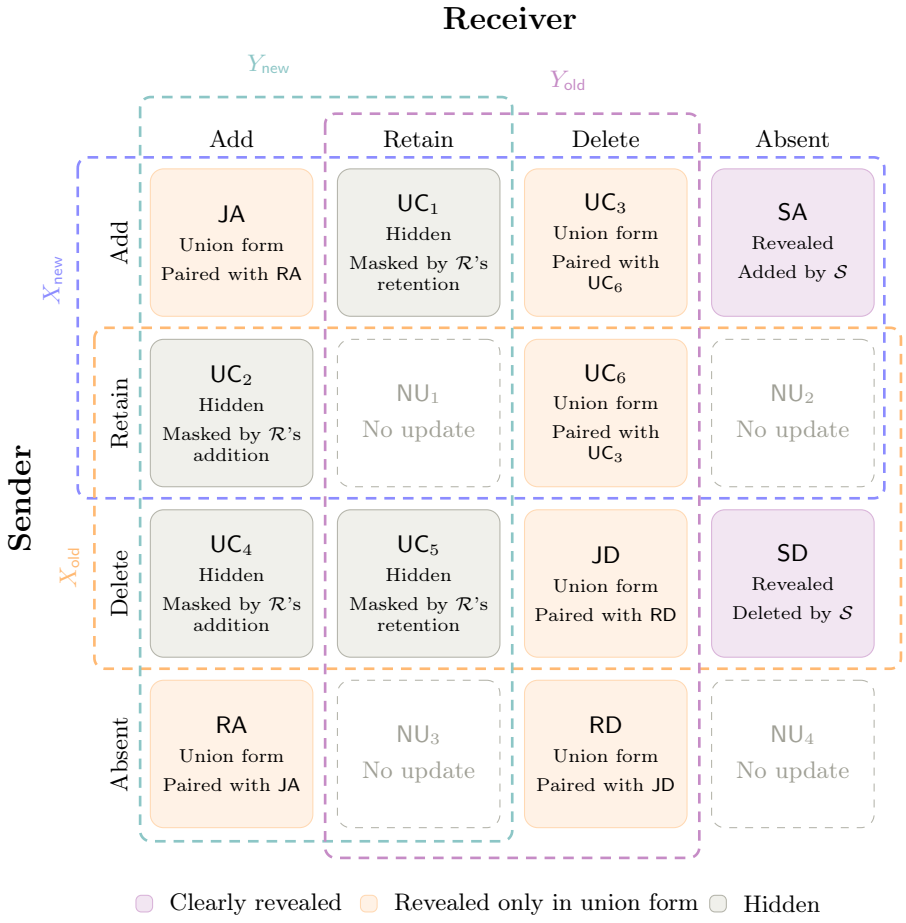

Fig. 2: Receiver's view of each update case under naïve PSU recomputation

– (SP 2: *Column 2*) All elements in the sets corresponding to the second column belong to both U_{old} and U_{new}. Thus, it is impossible to distinguish whether an element belongs to UC_1, NU_1, UC_5, or NU_3.
– (SP 3: *Column 3*) UC_3, UC_6, JD, and RD are all not contained in Y_{new}. Thus, from U_{old} and U_{new}, the receiver can distinguish $\mathsf{UC}_3 \cup \mathsf{UC}_6$ from $\mathsf{JD} \cup \mathsf{RD}$. However, it should remain indistinguishable whether an element in $\mathsf{UC}_3 \cup \mathsf{UC}_6$ belongs to UC_3 or UC_6. Likewise, it should remain indistinguishable whether an element in $\mathsf{JD} \cup \mathsf{RD}$ belongs to JD or RD.
– (SP 4: *Column 4*): SA and SD are inherently revealed by the sender's updates, while NU_2 and NU_4 are derivable from the receiver's view without affecting the union. These sets are obtained from the following relations:

$$\mathsf{SA} = (U_{\mathsf{new}} \setminus U_{\mathsf{old}}) \setminus Y_{\mathsf{new}}, \quad \mathsf{NU}_2 = (U_{\mathsf{old}} \setminus (Y_{\mathsf{new}} \cup Y_{\mathsf{old}})) \cap U_{\mathsf{new}}$$
$$\mathsf{SD} = (U_{\mathsf{old}} \setminus U_{\mathsf{new}}) \setminus Y_{\mathsf{old}}, \quad \mathsf{NU}_4 = (\mathcal{U} \setminus (Y_{\mathsf{old}} \cup Y_{\mathsf{new}})) \setminus (\mathsf{SA} \cup \mathsf{NU}_2 \cup \mathsf{SD})$$

where $\mathcal{U}$ denotes the universe set.

These security properties establish the leakage baseline that our protocol in Section 4 must not exceed.

3.3 A Worked Example

As an illustration of the classification, suppose the sender holds $X_{\mathsf{old}} = \{2, 4, 5, 7, 9, 10, 12\}$ and the receiver holds $Y_{\mathsf{old}} = \{1, 2, 5, 6, 7, 8, 11\}$, with union $U_{\mathsf{old}} = \{1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$. Both sides then update simultaneously: the sender adds $\{1, 3, 8, 13\}$ and deletes $\{2, 4, 5, 12\}$, while the receiver adds $\{4, 9, 13, 14\}$ and deletes $\{1, 5, 7, 11\}$. Thus the updated sets are $X_{\mathsf{new}} = \{1, 3, 7, 8, 9, 10, 13\}$ and $Y_{\mathsf{new}} = \{2, 4, 6, 8, 9, 13, 14\}$.

By the classification of Section 3.1, only six elements change the union: $3, 13, 14$ enter as $\mathsf{SA}, \mathsf{JA}, \mathsf{RA}$ and $5, 11, 12$ leave as $\mathsf{SD}, \mathsf{JD}, \mathsf{RD}$, respectively. The remaining affected elements $\{1, 2, 4, 7, 8, 9\}$ fall into $\mathsf{UC}_1, \ldots, \mathsf{UC}_6$ and remain in the union through at least one party. Hence, $U_{\mathsf{new}} = \{1, 2, 3, 4, 6, 7, 8, 9, 10, 13, 14\}$.

As predicted by Section 3.2, the information the receiver can infer from U_{old} and U_{new} is limited to the column level. For its own additions and deletions, the receiver learns that $\{13, 14\}$ entered and $\{5, 11\}$ left, but it cannot determine whether each element belong to the joint case (JA, JD) or the receiver-only case (RA, RD). The same ambiguity applies to the union-preserving siblings in these columns: $\mathsf{UC}_2 = \{9\}$ is indistinguishable from $\mathsf{UC}_4 = \{4\}$ and $\mathsf{UC}_3 = \{1\}$ is indistinguishable from $\mathsf{UC}_6 = \{7\}$, corresponding to (SP 1) and (SP 3). The retained cases $\mathsf{UC}_1 = \{8\}$ and $\mathsf{UC}_5 = \{2\}$ remain fully hidden, as in (SP 2), while only the sender-only changes $\mathsf{SA} = \{3\}$ and $\mathsf{SD} = \{12\}$ are revealed outright, as in (SP 4).

4 Our uPSU Protocol

In this section, we present our uPSU protocol that leverages existing PSI and tagged PSU protocols as building blocks. We first define the uPSU functionality $\mathcal{F}_{\mathsf{uPSU}}$, then present the protocol construction and its security proof against semi-honest adversaries.

Definition 5 (Ideal Functionality $\mathcal{F}_{\mathsf{uPSU}}$ of uPSU). *Let $(X_{\mathsf{old}}, X_{\mathsf{add}}, X_{\mathsf{del}})$ and $(Y_{\mathsf{old}}, Y_{\mathsf{add}}, Y_{\mathsf{del}}, U_{\mathsf{old}})$ be the inputs of the sender $\mathcal{S}$ and the receiver $\mathcal{R}$, respectively, where $|X_{\mathsf{old}}|, |Y_{\mathsf{old}}| \leq n$, $|X_{\mathsf{add}}|, |X_{\mathsf{del}}|, |Y_{\mathsf{add}}|, |Y_{\mathsf{del}}| \leq t$, and $U_{\mathsf{old}} = X_{\mathsf{old}} \cup Y_{\mathsf{old}}$. $\mathcal{R}$ receives the updated union $U_{\mathsf{new}} = X_{\mathsf{new}} \cup Y_{\mathsf{new}}$, where $X_{\mathsf{new}} = X_{\mathsf{add}} \cup (X_{\mathsf{old}} \setminus X_{\mathsf{del}})$ and $Y_{\mathsf{new}} = Y_{\mathsf{add}} \cup (Y_{\mathsf{old}} \setminus Y_{\mathsf{del}})$.*

4.1 Protocol Construction

We now present our uPSU protocol, whose complete description is shown in Figure 3 and which leverages existing building blocks in a novel way to achieve

Parameters:

- A sender $\mathcal{S}$ and a receiver $\mathcal{R}$
- Set size n and update threshold t.

Input:

- $\mathcal{S}$ inputs $(X_{\mathsf{old}}, X_{\mathsf{add}}, X_{\mathsf{del}})$, where $|X_{\mathsf{old}}| = n$, $|X_{\mathsf{add}}|, |X_{\mathsf{del}}| \leq t$, $X_{\mathsf{add}} \cap X_{\mathsf{old}} = \emptyset$ and $X_{\mathsf{del}} \subseteq X_{\mathsf{old}}$.
- $\mathcal{R}$ inputs $(Y_{\mathsf{old}}, Y_{\mathsf{add}}, Y_{\mathsf{del}}, U_{\mathsf{old}})$, where $|Y_{\mathsf{old}}| = n$, $|Y_{\mathsf{add}}|, |Y_{\mathsf{del}}| \leq t$, $Y_{\mathsf{add}} \cap Y_{\mathsf{old}} = \emptyset$ and $Y_{\mathsf{del}} \subseteq Y_{\mathsf{old}}$.

Output: $\mathcal{R}$ obtains $U_{\mathsf{new}} = X_{\mathsf{new}} \cup Y_{\mathsf{new}}$.

Protocol:

1. [Initialization]: $\mathcal{R}$ initiates U_{add} and U_{del} as empty sets.
2. [Sender's Update]:
   (a) $\mathcal{S}$ and $\mathcal{R}$ invoke the tagged PSU functionality $\mathcal{F}_{\mathsf{tPSU}}$:
      - $\mathcal{S}$ inputs X_{add} with tag $+$ and X_{del} with tag $-$, and $\mathcal{R}$ inputs $Y_{\mathsf{old}} \cup Y_{\mathsf{add}}$.
      - $\mathcal{R}$ obtains $\mathsf{SA} = X_{\mathsf{add}} \setminus (Y_{\mathsf{old}} \cup Y_{\mathsf{add}})$ with tag $+$ and $\mathsf{SD} = X_{\mathsf{del}} \setminus (Y_{\mathsf{old}} \cup Y_{\mathsf{add}})$ with tag $-$.

   (b) $\mathcal{R}$ updates $U_{\mathsf{add}} = U_{\mathsf{add}} \cup \mathsf{SA}$ and $U_{\mathsf{del}} = U_{\mathsf{del}} \cup \mathsf{SD}$.
3. [Receiver's Update]:
   (a) $\mathcal{R}$ computes $\mathsf{JA} \cup \mathsf{RA} = Y_{\mathsf{add}} \setminus (X_{\mathsf{old}} \setminus Y_{\mathsf{old}})$, where $X_{\mathsf{old}} \setminus Y_{\mathsf{old}} = U_{\mathsf{old}} \setminus Y_{\mathsf{old}}$. $\mathcal{R}$ then updates $U_{\mathsf{add}} = U_{\mathsf{add}} \cup (\mathsf{JA} \cup \mathsf{RA})$.
   (b) $\mathcal{S}$ and $\mathcal{R}$ invoke the PSI functionality $\mathcal{F}_{\mathsf{PSI}}$:
      - $\mathcal{S}$ inputs X_{new}, and $\mathcal{R}$ inputs Y_{del}.
      - $\mathcal{R}$ obtains $\mathsf{UC}_3 \cup \mathsf{UC}_6 = X_{\mathsf{new}} \cap Y_{\mathsf{del}}$.

   (c) $\mathcal{R}$ computes $\mathsf{JD} \cup \mathsf{RD} = Y_{\mathsf{del}} \setminus (\mathsf{UC}_3 \cup \mathsf{UC}_6)$ and then updates $U_{\mathsf{del}} = U_{\mathsf{del}} \cup (\mathsf{JD} \cup \mathsf{RD})$.
4. [Union Update]: $\mathcal{R}$ computes $U_{\mathsf{new}} = U_{\mathsf{add}} \cup (U_{\mathsf{old}} \setminus U_{\mathsf{del}})$.

Fig. 3: Our Updatable Private Set Union Protocol Π_{uPSU}

efficient updates. Below, we provide the design rationale and establish correctness.

Design Rationale. We begin with Eq. (1), which states that the updated union can be computed as $U_{\mathsf{new}} = U_{\mathsf{add}} \cup (U_{\mathsf{old}} \setminus U_{\mathsf{del}})$, where $U_{\mathsf{add}} = \mathsf{JA} \cup \mathsf{SA} \cup \mathsf{RA}$ and $U_{\mathsf{del}} = \mathsf{JD} \cup \mathsf{SD} \cup \mathsf{RD}$. Therefore, the task reduces to computing JA, SA, RA, JD, SD, and RD in a private manner that reveals no more information than full recomputation. (DI 1) and (DI 2) reduce these six cases to four set differences, two on the sender's side and two on the receiver's side, and (SP 1)–(SP 4) bound the leakage that each step may produce. We realize this specification using standard PSI and a tagged variant of PSU as building blocks.

- Tagged PSU for SA and SD: By (DI 1), SA and SD share the membership condition of non-membership in $Y_{\mathsf{old}} \cup Y_{\mathsf{add}}$. Most PSU protocols are constructed so that the sender obliviously transmits $X \setminus Y$ to the receiver as the core step of the protocol. Setting $X = X_{\mathsf{add}} \cup X_{\mathsf{del}}$ and $Y = Y_{\mathsf{old}} \cup Y_{\mathsf{add}}$ thus extracts $\mathsf{SA} \cup \mathsf{SD}$ in a single invocation. To preserve the opposite effects of SA and SD on the union, we employ a tagged PSU, which attaches a tag $+$ or $-$ to each transmitted message without any structural change to the underlying protocol. The resulting leakage matches (SP 4).

- Local computation of JA and RA by the receiver: By ($\mathsf{DI}\ 2$), $\mathsf{JA} \cup \mathsf{RA} = Y_{\mathsf{add}} \setminus X_{\mathsf{old}}$. The receiver already holds $X_{\mathsf{old}} \setminus Y_{\mathsf{old}} = U_{\mathsf{old}} \setminus Y_{\mathsf{old}}$ from the previous PSU execution, and since $Y_{\mathsf{add}} \cap Y_{\mathsf{old}} = \emptyset$, it holds that $Y_{\mathsf{add}} \setminus X_{\mathsf{old}} = Y_{\mathsf{add}} \setminus (X_{\mathsf{old}} \setminus Y_{\mathsf{old}})$. The receiver therefore computes $\mathsf{JA} \cup \mathsf{RA}$ locally, requiring no interaction with the sender. The union form of this output respects ($\mathsf{SP}\ 1$).
- PSI of X_{new} and Y_{del} for JD and RD: The receiver's deletions correspond to $\mathsf{JD} \cup \mathsf{RD} = Y_{\mathsf{del}} \setminus X_{\mathsf{new}}$ by ($\mathsf{DI}\ 2$), but no efficient protocol lets the receiver directly obtain $Y_{\mathsf{del}} \setminus X_{\mathsf{new}}$, i.e., the difference of its own set with the sender's set. We instead invoke PSI on X_{new} and Y_{del} to obtain $\mathsf{UC}_3 \cup \mathsf{UC}_6 = X_{\mathsf{new}} \cap Y_{\mathsf{del}}$, from which the receiver locally derives $\mathsf{JD} \cup \mathsf{RD} = Y_{\mathsf{del}} \setminus (\mathsf{UC}_3 \cup \mathsf{UC}_6)$. This construction respects ($\mathsf{SP}\ 3$): the two sets $\mathsf{JD} \cup \mathsf{RD}$ and $\mathsf{UC}_3 \cup \mathsf{UC}_6$ are distinguishable through the PSI output, while the internal composition of each union remains hidden.

Correctness. We show that the protocol correctly computes $U_{\mathsf{new}} = X_{\mathsf{new}} \cup Y_{\mathsf{new}}$. As discussed above, computing U_{new} reduces to obtaining U_{add} and U_{del}, which our protocol constructs as follows.

- $U_{\mathsf{add}} = \mathsf{SA} \cup \mathsf{JA} \cup \mathsf{RA}$: The tagged PSU in Step 2(a) returns SA with tag $+$, and the local computation in Step 3(a) returns $\mathsf{JA} \cup \mathsf{RA}$.
- $U_{\mathsf{del}} = \mathsf{SD} \cup \mathsf{JD} \cup \mathsf{RD}$: The tagged PSU in Step 2(a) returns SD with tag $-$, and the PSI in Step 3(b) returns $\mathsf{UC}_3 \cup \mathsf{UC}_6$, from which Step 3(c) derives $\mathsf{JD} \cup \mathsf{RD} = Y_{\mathsf{del}} \setminus (\mathsf{UC}_3 \cup \mathsf{UC}_6)$.

The receiver computes $U_{\mathsf{new}} = U_{\mathsf{add}} \cup (U_{\mathsf{old}} \setminus U_{\mathsf{del}})$ in Step 4, which equals the required updated union. The remaining cases are untouched by the protocol and thus remain in U_{old}, as required. Since both parties are semi-honest, each round outputs the correct union, so $U_{\mathsf{old}} = X_{\mathsf{old}} \cup Y_{\mathsf{old}}$ is inductively maintained across updates with no additional assumption.

4.2 Security Analysis

We prove the security of our protocol in the semi-honest model using the ideal/real paradigm with a sequence of hybrid experiments.

Security Model. We consider semi-honest adversaries who follow the protocol honestly but may attempt to learn additional information from the protocol transcript. Security is defined via simulation: for any real-world adversary, there exists an ideal-world simulator that produces an indistinguishable view using only the adversary's input and output. Our protocol operates in the $(\mathcal{F}_{\mathsf{tPSU}}, \mathcal{F}_{\mathsf{PSI}})$-hybrid model, making black-box use of ideal tagged PSU and PSI functionalities.

Theorem 1. *Protocol Π_{uPSU} securely realizes the uPSU functionality $\mathcal{F}_{\mathsf{uPSU}}$ in the $(\mathcal{F}_{\mathsf{tPSU}}, \mathcal{F}_{\mathsf{PSI}})$-hybrid model against semi-honest adversaries.*

Sketch of Proof. The key idea of our security proof is to construct a simulator that produces the outputs of $\mathcal{F}_{\mathsf{tPSU}}$ and $\mathcal{F}_{\mathsf{PSI}}$ using only the receiver's input $Y =$

$(Y_{\mathsf{old}}, Y_{\mathsf{add}}, Y_{\mathsf{del}}, U_{\mathsf{old}})$ and output U_{new}. For the tagged PSU output, the simulator first computes $\mathsf{JA} \cup \mathsf{RA} = Y_{\mathsf{add}} \setminus (X_{\mathsf{old}} \setminus Y_{\mathsf{old}})$, where $X_{\mathsf{old}} \setminus Y_{\mathsf{old}} = U_{\mathsf{old}} \setminus Y_{\mathsf{old}}$. Then, it derives $\mathsf{SA} = U_{\mathsf{new}} \setminus (U_{\mathsf{old}} \cup (\mathsf{JA} \cup \mathsf{RA}))$ with tag $+$ and $\mathsf{SD} = (U_{\mathsf{old}} \setminus U_{\mathsf{new}}) \setminus Y_{\mathsf{del}}$ with tag $-$. For the PSI output, the simulator computes $\mathsf{UC}_3 \cup \mathsf{UC}_6 = Y_{\mathsf{del}} \cap U_{\mathsf{new}}$, which equals $X_{\mathsf{new}} \cap Y_{\mathsf{del}}$ since any element $y \in Y_{\mathsf{del}}$ satisfies $y \notin Y_{\mathsf{new}}$, and thus $y \in X_{\mathsf{new}}$ if and only if $y \in U_{\mathsf{new}}$ for any $y \in Y_{\mathsf{del}}$.

Proof. We prove security separately for corrupted sender and corrupted receiver.

Case 1: Corrupted Sender. Since the sender receives no output in our protocol, i.e., $f_{\mathcal{S}}(X, Y) = \perp$, the simulator $\mathsf{Sim}_{\mathcal{S}}$ runs the honest sender strategy with input $X = (X_{\mathsf{old}}, X_{\mathsf{add}}, X_{\mathsf{del}})$ and uniformly random receiver messages. The simulation view is identically distributed to the real view, as the sender's view contains no information about the receiver's input and output.

Case 2: Corrupted Receiver. We prove security via a sequence of hybrid experiments, starting from the real protocol execution and ending in an ideal execution where the simulator uses only the receiver's input and output.

Let $\mathcal{A}$ be a real-world adversary corrupting the receiver with input $Y = (Y_{\mathsf{old}}, Y_{\mathsf{add}}, Y_{\mathsf{del}}, U_{\mathsf{old}})$ and output $f_{\mathcal{R}}(X, Y) = U_{\mathsf{new}}$.

Hybrid 0: This is the real protocol execution where the sender and receiver run the protocol honestly with inputs $X = (X_{\mathsf{old}}, X_{\mathsf{add}}, X_{\mathsf{del}})$ and $Y = (Y_{\mathsf{old}}, Y_{\mathsf{add}}, Y_{\mathsf{del}}, U_{\mathsf{old}})$, respectively. The receiver's view consists of:

- Output from tagged PSU: $\mathsf{SA} = X_{\mathsf{add}} \setminus (Y_{\mathsf{old}} \cup Y_{\mathsf{add}})$ with tag $+$ and $\mathsf{SD} = X_{\mathsf{del}} \setminus (Y_{\mathsf{old}} \cup Y_{\mathsf{add}})$ with tag $-$,
- Output from PSI: $\mathsf{UC}_3 \cup \mathsf{UC}_6 = X_{\mathsf{new}} \cap Y_{\mathsf{del}}$,
- Local computations.

Hybrid 1: In this hybrid, we replace the real execution of the tagged PSU with a simulated version. Given $Y = (Y_{\mathsf{old}}, Y_{\mathsf{add}}, Y_{\mathsf{del}}, U_{\mathsf{old}})$ and $U_{\mathsf{new}} = f_{\mathcal{R}}(X, Y)$, the simulator computes as follows:

1. $\mathsf{JA} \cup \mathsf{RA} = Y_{\mathsf{add}} \setminus (X_{\mathsf{old}} \setminus Y_{\mathsf{old}})$, where $X_{\mathsf{old}} \setminus Y_{\mathsf{old}} = U_{\mathsf{old}} \setminus Y_{\mathsf{old}}$.
2. $\mathsf{SA} = U_{\mathsf{new}} \setminus (U_{\mathsf{old}} \cup (\mathsf{JA} \cup \mathsf{RA}))$.
3. $\mathsf{SD} = (U_{\mathsf{old}} \setminus U_{\mathsf{new}}) \setminus Y_{\mathsf{del}}$.
4. The simulated output of $\mathcal{F}_{\mathsf{tPSU}}$ is SA with tag $+$ and SD with tag $-$.

By construction, $\mathsf{SA} = X_{\mathsf{add}} \setminus (Y_{\mathsf{old}} \cup Y_{\mathsf{add}})$ and $\mathsf{SD} = X_{\mathsf{del}} \setminus (Y_{\mathsf{old}} \cup Y_{\mathsf{add}})$, and these are the same as those in Steps 2 and 3, respectively. Thus, the distribution of *Hybrid* 1 is identical to *Hybrid* 0 since $\mathcal{F}_{\mathsf{tPSU}}$ reveals exactly SA with tag $+$ and SD with tag $-$.

Hybrid 2: In this hybrid, we additionally replace the real PSI execution with a simulated version. The simulator computes as follows:

1. Perform all simulations from *Hybrid* 1.
2. Determine the set $\mathsf{UC}_3 \cup \mathsf{UC}_6 = Y_{\mathsf{del}} \cap U_{\mathsf{new}}$.
3. The simulated output of $\mathcal{F}_{\mathsf{PSI}}$ is $\mathsf{UC}_3 \cup \mathsf{UC}_6$.

By construction, $\mathsf{UC}_3 \cup \mathsf{UC}_6 = X_{\mathsf{new}} \cap Y_{\mathsf{del}}$. Since any element $y \in Y_{\mathsf{del}}$ satisfies $y \notin Y_{\mathsf{new}}$, it is in X_{new} if and only if $y \in U_{\mathsf{new}}$. As $\mathcal{F}_{\mathsf{PSI}}$ reveals exactly $\mathsf{UC}_3 \cup \mathsf{UC}_6$, the distribution of *Hybrid* 2 is identical to *Hybrid* 1.

Hybrid 3: This hybrid represents the simulated view in the ideal world. The simulator $\mathsf{Sim}_{\mathcal{R}}(1^\kappa, Y, U_{\mathsf{new}})$ performs exactly the computations described in *Hybrid* 2. By definition, the distribution of *Hybrid* 3 is identical to *Hybrid* 2.

By the sequence of hybrids, we have

$$\begin{aligned}\{\mathsf{View}^{\Pi}_{\mathcal{R}}(X, Y)\} &= \textit{Hybrid } 0 \equiv \textit{Hybrid } 1 \equiv \textit{Hybrid } 2 \equiv \textit{Hybrid } 3 \\ &= \{\mathsf{Sim}_{\mathcal{R}}(1^\kappa, Y, f_{\mathcal{R}}(X, Y))\},\end{aligned}$$

which shows that $\mathsf{Sim}_{\mathcal{R}}(1^\kappa, Y, f_{\mathcal{R}}(X, Y)) \overset{c}{\approx} \mathsf{View}^{\Pi}_{\mathcal{R}}(X, Y)$. Combined with the simulation for corrupted sender, this completes the proof. □

5 Performance Evaluation

In this section, we evaluate the communication cost and running time of our protocol against JSZG24 [18], TBZ+25 [30], and KLS26 [19] across the various network settings and set sizes.

5.1 Experimental Setup

Choice of Building Blocks. We implement our protocol in C++, and the source code is available in the GitHub repository[1]. Since our construction is generic in the sense that it can be realized by any tagged PSU and PSI protocols, we have several candidates for building blocks. Among them, we selected KLS26 [19] and RR22 [28] as the PSU and PSI building blocks, respectively, for computation efficiency. KLS26 is built on oblivious pseudorandom function (OPRF), circuit-PSI, permute and share (P&S), and oblivious transfer (OT). Its OT-based transfer step directly accommodates our tagged PSU variant: the sender appends a tag bit to each transmitted message, namely $m\|0$ for additions and $m\|1$ for deletions, requiring no structural modification to the underlying protocol. Although KLS26 is primarily designed for the balanced setting, it naturally extends to the unbalanced setting by adjusting the circuit-PSI parameters. RR22 is one of the state-of-the-art PSI protocols with linear communication complexity, which is critical for the receiver's update phase. By plugging in building blocks for the unbalanced setting, our generic construction is also able to achieve communication efficiency.

We compare our protocol with JSZG24 [18], TBZ+25 [30], and KLS26 as full-recomputation baseline PSU protocols. For fair comparison, we obtained the original source code of JSZG24 from the GitHub repository[2], TBZ+25 from Zenodo[3], and KLS26 from the GitHub repository[4]. We additionally ran CKY25 [11]

[1] https://github.com/SeongbongChoi/updatable-PSU
[2] https://github.com/yanxue820/SecurePSU.git
[3] https://doi.org/10.5281/zenodo.14725816
[4] https://github.com/yonghaason/volepsi

Table 1: Performance of Our Protocol for Various Input and Update Sizes

Protocols	n											
	2^{16}				2^{18}				2^{20}			
	t	Comm. (MB)	Runtime (s) LAN	Runtime (s) WAN	t	Comm. (MB)	Runtime (s) LAN	Runtime (s) WAN	t	Comm. (MB)	Runtime (s) LAN	Runtime (s) WAN
JSZG24	–	137.42	4.32	32.49	–	577.73	11.76	112.75	–	2430.47	43.44	455.68
TBZ+25		16.51	9.23	13.46		64.42	35.04	46.92		258.75	139.29	181.79
KLS26		13.38	3.80	8.69		49.92	16.26	26.34		199.22	68.38	100.87
Ours	2^{8}	4.27	0.54	4.09	2^{10}	11.71	0.98	6.0	2^{12}	41.12	3.07	13.14
	2^{9}	4.46	0.55	4.15	2^{11}	12.08	1.06	6.12	2^{13}	43.77	3.56	14.07
	2^{10}	4.67	0.58	4.25	2^{12}	12.86	1.21	6.38	2^{14}	46.35	4.46	15.40
	2^{11}	5.04	0.66	4.36	2^{13}	14.51	1.63	7.10	2^{15}	56.38	6.40	18.74
	2^{12}	5.81	0.80	4.62	2^{14}	17.09	2.49	8.37	2^{16}	68.19	10.83	24.51
	2^{13}	7.46	1.22	5.35	2^{15}	24.30	4.26	11.32	2^{17}	91.90	20.20	35.97

only as a reference, obtaining its source code from the GitHub repository[5] and adding a network layer that the original implementation lacks.

Implementation Setup. All experiments were conducted on a virtual machine with 4 cores and 8 threads running Ubuntu 20.04 LTS. The host system consists of an Intel Xeon Silver 4215 CPU at 3.2 GHz and 128 GB of RAM, operating on Windows Server 2019. However, we configured all experiments to use only a single core and single thread. Each reported measurement represents the average of 10 independent executions. We evaluate our protocol in two network settings: LAN (10Gbps bandwidth, 0.2ms RTT) and WAN (50Mbps, 80ms RTT), emulated using the Linux TC tool. We set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$.

5.2 Experimental Results

Table 1 presents the performance of our protocol across various set sizes and update thresholds. We evaluate update ratios ranging from $\frac{2t}{n} \approx 0.78\%$ to 25% of the original set size, covering a substantially broader range than prior updatable PSI works [4, 24, 25, 3], which mainly evaluate update sizes below 2%. For $n = 2^{16}$, our protocol achieves a 3.5–8.0× speedup in LAN over JSZG24, 7.6–17.1× over TBZ+25, and 3.1–7.0× over KLS26, with 18.4–32.2×, 2.2–3.9×, and 1.8–3.1× communication reductions, respectively. These gains persist at larger set sizes: for $n = 2^{20}$ and $t = 2^{12}$, our protocol completes in 3.07 seconds in LAN, achieving 14.1×, 45.4×, and 22.2× speedup over JSZG24, TBZ+25, and KLS26 with 59.1×, 6.3×, and 4.8× communication reductions, respectively. Even at $t = 2^{17}$, corresponding to an update ratio of 25%, our protocol maintains a 2.2× speedup over JSZG24, 6.9× over TBZ+25, and 3.4× over KLS26. The running time and communication cost of our protocol grow linearly with the update size, whereas JSZG24, TBZ+25, and KLS26 incur constant cost regardless of update

[5] https://github.com/heewon-chung/UD-PSU

Table 2: Performance Breakdown of Subprotocols for Representative Configurations

Subprotocol		(n,t)								
		$(2^{16},2^{12})$			$(2^{18},2^{12})$			$(2^{20},2^{14})$		
		Comm. (MB)	Runtime (s) LAN	WAN	Comm. (MB)	Runtime (s) LAN	WAN	Comm. (MB)	Runtime (s) LAN	WAN
Sender's Update	OPRF	0.61	0.08	0.75	0.75	0.09	0.79	1.27	0.16	0.96
	Circuit-PSI	2.64	0.29	1.87	8.60	0.80	3.44	32.36	3.23	9.91
	P&S	0.11	0.06	0.45	0.18	0.09	0.45	0.39	0.24	0.54
	OT	0.16	0.07	0.43	0.42	0.08	0.57	1.41	0.17	0.84
	Subtotal	3.52	0.50	3.50	9.95	1.06	5.25	35.43	3.80	12.25
Receiver's Update	PSI	1.15	0.06	0.72	2.91	0.08	1.03	10.92	0.14	2.60
	Local	–	0.02	0.03	–	0.08	0.10	–	0.52	0.55
	Subtotal	1.15	0.08	0.75	2.91	0.16	1.13	10.92	0.66	3.15
Total		4.67	0.58	4.25	12.86	1.21	6.38	46.35	4.46	15.40

size. For reference, CKY25 [11], the only prior PSU protocol supporting updates, runs in a delegated multi-party model and is therefore not directly comparable. In our setup it took over 670 s at $n = 2^{16}$, which is $\approx 1400\times$ slower than our protocol.

5.3 Performance Breakdown

We present a detailed breakdown of the performance of our protocol by component for a representative configuration in Table 2. As described in Section 5.1, we instantiate our protocol with KLS26 for (tagged) PSU and RR22 for PSI. KLS26 consists of OPRF, circuit-PSI, P&S, and OT components.

The sender's update phase dominates the overall cost of the protocol. For $(n,t) = (2^{16},2^{10})$, this phase accounts for 3.52 MB out of 4.67 MB in total communication, corresponding to 75.4% of the overall communication cost. It also consumes 0.50 seconds out of a total running time of 0.58 seconds in the LAN setting, representing 86.3% of the total execution time. Within this phase, circuit-PSI constitutes the primary bottleneck, incurring 2.64 MB of communication and 0.29 seconds of running time.

In contrast, the receiver's update phase is relatively lightweight. This phase consists of a PSI invocation and local computations. For the same configuration, the receiver's update requires only 1.15 MB in communication, accounting for 24.6% of the total, and 0.08 seconds in the LAN setting, which represents 13.7% of the total running time. The local computation that determines receiver additions and updates the union is negligible, taking only 0.02 seconds. These proportions remain consistent as the configuration scales to larger set sizes: circuit-PSI continues to dominate the cost of the sender's update phase, while the receiver's update phase remains efficient.

In the WAN setting, network latency alters the relative performance of individual components. Communication-intensive subprotocols such as circuit-PSI

experience more pronounced increases in running time than computation-heavy components due to added latency. For example, for $(n, t) = (2^{16}, 2^{10})$, the running time of circuit-PSI increases from 0.29 seconds in the LAN setting to 1.87 seconds in the WAN setting, whereas local computation remains nearly constant. This analysis confirms that circuit-PSI is the main performance bottleneck and thus the most promising target for future optimization. Nevertheless, even in its current form, our protocol already demonstrates significant advantages over existing approaches to updatable PSU.

6 Conclusion and Open Problem

We presented the first updatable PSU (uPSU) protocol for the standard two-party setting that enables efficient incremental updates without full recomputation of the entire union from scratch. Our approach is based on a systematic classification of all possible update scenarios into 12 disjoint cases, which reveals that only 6 cases actually modify the union. Leveraging this insight, we presented a generic construction for uPSU that uses existing PSI and a tagged variant of PSU as building blocks, allowing any secure instantiation of these primitives to be plugged in. Our experimental results demonstrated substantial advantages over existing approaches. Compared with full-recomputation PSU baselines (JSZG24, TBZ+25, and KLS26), our protocol achieved a 14.1–45.4× speedup in the LAN setting while reducing communication by 4.8–59.1×. The only prior PSU protocol supporting update, CKY25, operates in a delegated multi-party model and is not a directly comparable two-party baseline.

While our protocol substantially improves over full recomputation, it still leaves room for further improvement. In particular, both communication and computation costs still depend on the original set sizes rather than purely on the update size. Although the update sets themselves have size $O(t)$, the underlying cryptographic protocols require configuration based on the larger set sizes. Specifically, in the sender's update phase, the circuit-PSI protocol requires configuration for the receiver's set size $n + t$ and causes both communication and computation to scale with $O(n)$. Similarly, in the receiver's update phase, the PSI step processes the sender's full updated set X_{new} of size n. As shown in Table 1, for the same update threshold $t = 2^{10}$, our protocol required 4.67 MB for $n = 2^{16}$, but 11.71 MB for $n = 2^{18}$. This demonstrates the dependency on the original set size. An ideal uPSU protocol would achieve both $O(t)$ communication and $O(t)$ computation costs, independent of the original set sizes. One potential direction is to apply an unbalanced PSU protocol, which is specifically designed for asymmetric set sizes and whose communication cost depends on the smaller set size. However, since unbalanced PSU constructions typically rely on homomorphic encryption, they introduce substantial computational overhead despite the potential communication savings. While our protocol performed significantly better than full recomputation, the elimination of this n-dependency remains an important open problem.

Acknowledgments. The authors would like to thank the anonymous reviewers for their helpful comments. This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00399491, 100%).

References

1. A. Abadi, C. Dong, S. J. Murdoch, and S. Terzis. Multi-party updatable delegated private set intersection. In *FC 2022*, volume 13411 of *Lecture Notes in Computer Science*, pages 100–119, 2022.
2. A. Agarwal, D. Cash, M. George, S. Kamara, T. Moataz, and J. Singh. Updatable private set intersection from structured encryption. *IACR Communications in Cryptology*, 2(4):25, 2026.
3. F. Alborch, T. Chauvier, A. Faonio, A. Fontaine, F. Karakoç, A. Küpçü, C. Malek, and M. Önen. Updatable private set intersection and beyond: Efficient constructions via circuit private set intersection. In *ACNS 2026*, Lecture Notes in Computer Science, 2026. Available at https://eprint.iacr.org/2025/2147.
4. S. Badrinarayanan, P. Miao, X. Shi, M. Tromanhauser, and R. Zeng. Updatable private set intersection revisited: Extended functionalities, deletion, and worst-case complexity. In *ASIACRYPT 2024*, volume 15489 of *Lecture Notes in Computer Science*, pages 200–233, 2024.
5. S. Badrinarayanan, P. Miao, and T. Xie. Updatable private set intersection. *Proceedings on Privacy Enhancing Technologies*, 2022(2):378–406, 2022.
6. M. Blanton and E. Aguiar. Private and oblivious set and multiset operations. In *ASIACCS 2012*, pages 40–41, 2012.
7. P. Buddhavarapu, A. Knox, P. Mohassel, S. Sengupta, E. Taubeneck, and V. Vlaskin. Private matching for compute. Cryptology ePrint Archive, Paper 2020/599, 2020.
8. M. Burkhart, M. Strasser, D. Many, and X. A. Dimitropoulos. SEPIA: privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security 2010*, pages 223–240, 2010.
9. G. R. Chandran, T. Schneider, M. Stillger, and C. Weinert. Concretely efficient private set union via circuit-based PSI. In *ACM ASIA CCS 2025*, pages 149–162, 2025.
10. Y. Chen, M. Zhang, C. Zhang, M. Dong, and W. Liu. Private set operations from multi-query reverse private membership test. In *PKC 2024*, volume 14603 of *Lecture Notes in Computer Science*, pages 387–416, 2024.
11. H. Chung, M. Kim, and P. C. Yang. Multiparty delegated private set union with efficient updates on outsourced datasets. *IEEE Transactions on Dependable and Secure Computing*, 22(2):903–918, 2025.
12. A. Davidson and C. Cid. An efficient toolkit for computing private set operations. In *ACISP 2017*, volume 10343 of *Lecture Notes in Computer Science*, pages 261–278, 2017.
13. K. B. Frikken. Privacy-preserving set union. In *ACNS 2007*, volume 4521 of *Lecture Notes in Computer Science*, pages 237–252, 2007.
14. G. Garimella, P. Mohassel, M. Rosulek, S. Sadeghian, and J. Singh. Private set operations from oblivious switching. In *PKC 2021*, volume 12711 of *Lecture Notes in Computer Science*, pages 591–617, 2021.

15. S. D. Gordon, C. Hazay, P. H. Le, and M. Liang. More efficient (reusable) private set union. Cryptology ePrint Archive, Paper 2022/713, 2022.
16. K. Hogan, N. Luther, N. Schear, E. Shen, D. Stott, S. Yakoubov, and A. Yerukhimovich. Secure multiparty computation for cooperative cyber risk assessment. In *IEEE SecDev 2016*, pages 75–76, 2016.
17. Y. Jia, S. Sun, H. Zhou, J. Du, and D. Gu. Shuffle-based private set union: Faster and more secure. In *USENIX Security 2022*, pages 2947–2964, 2022.
18. Y. Jia, S. Sun, H. Zhou, and D. Gu. Scalable private set union, with stronger security. In *USENIX Security 2024*, 2024.
19. J. Kim, H. T. Lee, and Y. Son. Private set operations from circuit-based PSI. In *SAC 2026*, pages 1857–1866, 2026.
20. Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas. Private set intersection for unequal set sizes with mobile applications. *Proceedings on Privacy Enhancing Technologies*, 2017(4):177–197, 2017.
21. L. Kissner and D. X. Song. Privacy-preserving set operations. In *CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257, 2005.
22. V. Kolesnikov, M. Rosulek, N. Trieu, and X. Wang. Scalable private set union from symmetric-key techniques. In *ASIACRYPT 2019*, volume 11922 of *Lecture Notes in Computer Science*, pages 636–666, 2019.
23. A. K. Lenstra and T. Voss. Information security risk assessment, aggregation, and mitigation. In *ACISP 2004*, volume 3108 of *Lecture Notes in Computer Science*, pages 391–401, 2004.
24. G. Ling, P. Tang, S. Sun, and W. Qiu. Efficient updatable PSI from asymmetric PSI and PSU. *IEEE Transactions on Information Forensics and Security*, 21:2626–2640, 2026.
25. J. Liu, P. Miao, M. Rosulek, X. Shi, and J. Wang. Updatable private set intersection from symmetric-key techniques. In *EUROCRYPT 2026*, Lecture Notes in Computer Science, pages 376–407, 2026.
26. L. Piske and N. Trieu. Is PSI really faster than psu? achieving efficient PSU with invertible bloom filters. In *EUROCRYPT 2026*, Lecture Notes in Computer Science, pages 438–467, 2026.
27. S. Pu, J. Gao, and N. Trieu. Malicious private set union with two-sided output. In *EUROCRYPT 2026*, Lecture Notes in Computer Science, pages 408–437, 2026.
28. S. Raghuraman and P. Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In *ACM CCS 2022*, pages 2505–2517, 2022.
29. S. Ramanathan, J. Mirkovic, and M. Yu. BLAG: improving the accuracy of blacklists. In *NDSS 2020*, 2020.
30. B. Tu, Y. Bai, C. Zhang, Y. Cao, and Y. Chen. Fast enhanced private set union in the balanced and unbalanced scenarios. In *USENIX Security 2025*, pages 3437–3456, 2025.
31. B. Tu, Y. Chen, Q. Liu, and C. Zhang. Fast unbalanced private set union from fully homomorphic encryption. In *ACM CCS 2023*, pages 2959–2973, 2023.
32. R. Wang, J. Zhou, Z. Cao, X. Dong, and K. R. Choo. Updatable private set intersection with forward privacy. *IEEE Transactions on Information Forensics and Security*, 19:8573–8586, 2024.
33. C. Zhang, Y. Chen, W. Liu, L. Peng, M. Hao, A. Wang, and X. Wang. Unbalanced private set union with reduced computation and communication. In *ACM CCS 2024*, pages 1434–1447, 2024.
34. C. Zhang, Y. Chen, W. Liu, M. Zhang, and D. Lin. Linear private set union from multi-query reverse private membership test. In *USENIX Security 2023*, pages 337–354, 2023.