# Updatable Private Set Intersection and Beyond: Efficient Constructions via Circuit PSI

Ferran Alborch[1], Tom Chauvier[1], Antonio Faonio[1], Alexandre Fontaine[1], Ferhat Karakoç[2], Alptekin Küpçü[3], Camille Malek[1], and Melek Önen[1]

[1] EURECOM, Sophia Antipolis, France
{ferran.alborch,antonio.faonio,melek.onen}@eurecom.fr
[2] Ericsson Research, İstanbul, Türkiye
ferhat.karakoc@ericsson.com
[3] Koç University, İstanbul, Türkiye
akupcu@ku.edu.tr

**Abstract.** Private Set Intersection (PSI) has been widely studied, deployed, and demonstrated on static datasets. In this work, we investigate the problem of designing efficient and secure updatable PSIs in the honest-but-curious model by adopting the approach of executing a small number of PSIs over smaller sets instead of one PSI over the entire updated sets. We first identify that existing constructions suffer from privacy leakages and further propose to mitigate them thanks to the use of circuit PSIs, which are variants of PSI protocols that output the secret shares of the intersection instead of outputting the resulting intersection, combined with secure shuffling when needed. We construct a generic framework for PSI over updated sets and show that this framework can easily be extended to a protocol that outputs the cardinality of the intersection instead of the intersection itself.

**Keywords:** Private set intersection · Dynamic datasets · circuit-PSI

## 1 Introduction

Private Set Intersection (PSI) enables two mutually distrusting parties, each of them holding some private datasets, to compare these datasets and disclose only their common records, and nothing more. This functionality excels in various use cases, such as mobile private contact discovery [26, 28, 37], privacy-preserving contact tracing [8, 50, 52], online advertising measurement [22], and privacy breach detection [40]. Researchers have investigated various challenges such as scalability [29], balanced and unbalanced settings with respect to the size of the datasets or the resources of the actual parties [48], the extension to the multi-party setting [34], or additional features whereby instead of revealing the actual intersection, the protocol only reveals the output of a function over this intersection (cardinality for statistical analysis, for example) [27].

One aspect that has not yet been investigated deeply is the case where datasets are not static, i.e. datasets are modified over time. Indeed, real datasets

are frequently updated as new data is generated or collected and some data might become useless or have expired. In this context, the computed intersections or other further computations over these intersections can become obsolete rapidly. While there exist some solutions that tackle this problem by taking advantage of the existing output and therefore not re-executing the PSI or circuit-PSI protocol from scratch [2, 4, 5, 35, 51], we observe that these solutions still suffer from privacy leakages (we refer to Figure 1 and Section 2 for more details).

**Our Contributions.** In this paper, we identify and study these leakages, which arise when one considers both additions of new elements and deletions of old elements in the datasets, and further propose new constructions for PSI and cardinality of PSI over updated sets in the honest-but-curious model. Inspired by Badrinarayanan et.al. [4], our constructions rely on the splitting of the datasets into smaller ones and performing some operations over them separately, before obliviously re-unifying the intermediate outputs. Apart from asymptotic efficiency, an important advantage of our solution is that it supports the (one-sided) client-server setting where only the client receives the ultimate output (the actual new intersection).

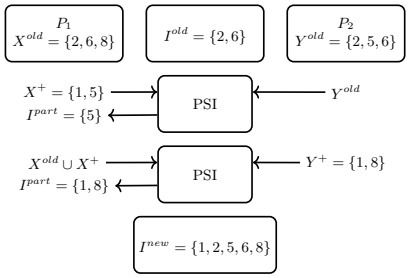
## 2 Problem Statement

PSI protocols are two-party protocols allowing parties $P_1$ and $P_2$, each owning (static) sets $X, Y$, respectively, to compute the intersection $X \cap Y$ without leaking any extra information besides the cardinality of at least one of the input sets.

For some applications, however, outputting the intersection itself might be too much. For example, if the use case requires computing the cardinality of the intersection only (e.g., when computing counting queries over shared data), outputting the elements of the intersection could be considered a leakage. In this case, we can use Cardinality-PSI protocols.

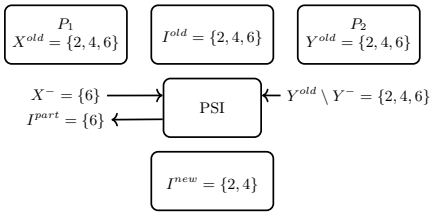
Other use cases require computing arbitrary statistics over the intersection. In general, the parties may want to compute an arbitrary function $f(X \cap Y)$. A particular kind of PSI is then used, called Circuit-PSI, where the parties obtain secret shares of the intersection. These shares can then be used within a two-party computation (2PC) protocol to obtain the intended output of $f$ over the intersection.

**Updatable PSI (uPSI).** Updatable PSI, and its variants such as updatable cardinality-PSI, or updatable circuit-PSI, protocols deal with dynamic sets being updated through time, and their intersection (or an appropriate function over it) being updated along with it. The main goal of updatable PSI is to speed up the computation of PSI on updated sets by leveraging the knowledge of the intersection (or the cardinality, or the secret shares) from the previous sets. The underlying assumption is that the updates are orders of magnitude smaller than the full dataset. For a more detailed review of the related work we refer to Section 3.




$P_1$ knows $P_2$ added 1 in the last update and 5 was in $Y^{old}$.

(a) Leakage from partial updates (we assume $X^- = Y^- = \emptyset$).




$P_1$ knows $P_2$ still has 6 in its database.

(b) Leakage from the size of the update (we assume $X^+ = Y^+ = Y^- = \emptyset$).

Fig. 1: Representation of leakages in updatable private set intersection.

Typical solutions to updatable PSI or updatable cardinality-PSI, including our solutions, split the sets into significantly smaller ones (such as the set of added or removed elements), and execute private set intersections and some other operations over these smaller sets [4, 5, 35, 51]. The difference between the solutions in the related work and ours is how these sets are defined and used to implement the operations on them.

While with this approach (and similar approaches) there is a significant performance gain, we observe that the independent execution of multiple PSIs or cardinality-PSIs among smaller sets results in some unauthorized disclosure of either elements of the set or some information about the cardinality of sets.

Indeed, consider the example in Figure 1. We denote the old sets with $X^{old}, Y^{old}$, the newly-added elements with $X^+, Y^+$, and the elements removed from the respective sets with $X^-, Y^-$. Thus we have new sets $X^{new} = (X^{old} \setminus X^-) \cup X^+$ and $Y^{new} = (Y^{old} \setminus Y^-) \cup Y^+$. $I^{old}$ and $I^{new}$ denote the old and new intersections. The protocol in Figure 1a computes the new intersection by computing the union of $X^+ \cap Y^{old}$ and $X^{new} \cap Y^+$. For simplicity, the protocol works when we assume $X^-$ and $Y^-$ are empty. In the figure, we consider the case where both $P_1$ and $P_2$ are adding the same element (element 1 in the example) and $P_1$ is adding an element $P_2$ already had in its old set (element 5 in the example). By receiving the intermediate intersections in cleartext, $P_1$ discovers that $P_2$ added element 1, and element 5 was already in $P_2$'s old dataset, whereas, in an ideal execution of the updated intersection, these should be hidden. In particular, $P_1$ can distinguish the case where $P_2$ starts with $\{2, 1, 6\}$ and adds $\{5, 8\}$ from the case where $P_2$ starts with $\{2, 5, 6\}$ and adds $\{1, 8\}$. This information, which is analogous in the case of deleted elements, then becomes an unauthorized leakage. The work in [51] suffers of this form of leakage.

To illustrate such a leakage with a real-life use case, we consider the use case of privacy-preserving contact tracing during the COVID-19 pandemic [8,50]. The list of users that a user is in contact with is always being updated with time. Hence, when using the naïve PSI with smaller sets, the application can leak the

list of users infected *at the same time*, which adds the additional information of *when* the users were infected with respect to the intended leakage of *only the infected users*.

Therefore, the outputs of the intermediate operations executed on smaller sets need to be protected. The natural approach for this is to use Circuit-PSI, due to the inherent protection provided by its secret-shared output. This observation has already been made, either implicitly or explicitly, in prior work such as [2, 4, 35].

Nevertheless, we identify that this new proposal still suffers from two additional categories of leakages: Namely, (i) the leakage of the size of the input sets to (the smaller instances of) circuit-PSIs and (ii) the leakage of reconstructed outputs from (the smaller instances of) circuit-PSIs.

For the former form of leakage, consider the example in Figure 1b. When $P_1$ removes an element of the old intersection from its database (element 6 in the example) while $P_2$ does not remove anything (i.e., $|Y^-| = 0$), $P_1$ can infer that this element is still in $P_2$'s database due to the knowledge of that $|Y^{old}| = |Y^{old} \setminus Y^-|$. This information, which is analogous in the case of added elements, becomes an unauthorized leakage. For example, in the use case of private contact discovery in messaging applications, PSI helps new users discover other users in their contact list who employ the same messaging application. As the contact list is regularly updated, the previously described leakage (for example, the size of removed contacts) would allow a contact being blocked by a user to distinguish whether he/she was blocked or the user left the app. To address this leakage, we propose to pad the sets for the circuit-PSIs and obtain a fixed size of $\alpha$ elements for each of them.

For the latter form of leakage, we notice that to compute the new intersection, one of the parties needs to reconstruct using the output of the (smaller instances of) circuit-PSIs. As previously mentioned using Figure 1a, such outputs result in some unauthorized leakage. In the context of cardinality-PSI, this does not pose a problem because only the final output is reconstructed (hence intermediate cardinalities are not leaked). In the context of updatable PSI though, the elements themselves need to be revealed. The party reconstructing intermediate outputs should not be able to discover which element is the output of which circuit-PSI. Note that the exact information leaked by the naïve reconstruction of shares is the same as when not using circuit-PSI. As such, the same previously described examples on the discovery of infected people apply here as well.To address this leakage, we propose to use an oblivious permutation, namely to randomly shuffle the secret shared values, before the reconstruction of the actual elements.

## 3 Related Work

**Private Set Intersection.** The first PSI construction was proposed by Meadows in [38], based on the Diffie-Hellman key exchange, and protocols have been since proposed based on a plethora of different cryptographic primitives, like Cuckoo hashing and oblivious transfer [33, 41, 43], polynomials over finite fields

[18, 19, 25], homomorphic encryption [15, 16], or oblivious key-value stores and vector oblivious linear evaluation [42, 44, 45]. A PSI protocol is defined as *one-sided* if only one of the parties obtains the intersection, and *two-sided* if both of them obtain the output. It is clear that in the honest-but-curious setting one-sided protocols imply two-sided protocols but the inverse is not true. While PSI protocols usually assume both parties have similar computing power and set sizes, some PSI protocols [23,36,48] are *unbalanced* or *asymmetric*: a (potentially client $P_1$ with small set $X$ and a powerful server $P_2$ with very large set $Y$. The objective is then to achieve computational or communication cost complexity linear with the size of $X$ and sublinear with respect to the size of $Y$.

**Circuit-Private Set Intersection.** Circuit PSI, first introduced by Huang et al. in [27], is a variant of PSI where instead of outputting the intersection, it outputs the output of a circuit executed over this intersection (without leaking the intersection itself). Some of the most common circuits are *cardinality*, which outputs only the size of the intersection, *payload sum*, where each element has an associated payload and the protocol outputs the sum of those related to the elements of the intersection, and *threshold*, where the intersection is output only if its size is greater than some predetermined threshold.

Recent Circuit PSI constructions deliver a generic output of secret shares of the intersection [3, 12, 30, 45, 48, 53] to be further used in any upcoming circuit, with linear computation and communication. The most generalized version encompassing the rest is that of [31], where the output given to $P_1$ is any strings $d_i^0, d_i^1$ where the first one is received if the element $i$ is not in the intersection and otherwise the second is received.

**Updatable Private Set Intersection**. The study of private set intersection for updatable sets involves two parties computing the intersection on their respective dynamic private sets. The key problem to solve is to find a protocol that can be run over the updates of the respective sets to update the (previously computed) intersection of the old sets in such a way that it is more efficient than re-running a standard PSI protocol over the updated sets.

Updatable PSI was first explored by Kiss et al. [32] in the setting of mobile applications by developing a solution with costly pre-computation, which can be updated. Unfortunately, the paper does not provide any formal analysis of the leakage when the datasets are updated. Abadi et al. [1] propose a solution for the multi-party case by delegating the computation to a server, which becomes impractical in a setting considering only two parties.

The first work to formalize updatable PSI and give concrete security definitions as well as an analysis of the leakage is by Badrinayan et al. [5]. They propose two one-sided solutions based on the DDH assumption, one that supports arbitrary inserts, and one for arbitrary inserts along with "weak deletion". Here weak deletion implies that elements inserted in the latest $t$ days can be deleted (where $t$ is a parameter). Their constructions also leak the size of the updates per day and the size of intermediate intersections.

In a follow-up work [4], they give one-sided solutions for arbitrary deletions as well as tackling some extended functionalities like cardinality or payload sum

Table 1: Comparison of updatable PSI schemes with extended functionalities, where $\alpha$ denotes the size of the updates and $n$ the size of the sets.

| | Func. | Updates | Communication Complexity | Output |
|---|---|---|---|---|
| [5] | PSI | Add | $O(\alpha \cdot \log n)$ | One-sided |
| [4] | | | $O(\alpha \cdot \log n)$ | One-sided |
| [4] | | Add & Del | $O(\alpha \cdot \log^2 n)$ | One-sided |
| [51] | | | $O(\alpha \cdot \log n)$[†] | Two-sided |
| [2] | | | $O(\alpha \cdot \log n^2)$ | Two-sided |
| [35] | | | $O(\alpha \cdot \log n + \sqrt{n})$ | Two-sided |
| Section 5 | | | $O(\alpha \cdot \log n + \alpha \cdot \log \alpha)$ | One-sided |
| [4] | Card. | Add | $O(\alpha \cdot \log n)$ | One-sided |
| [4] | | Add & Del | $O(\alpha \cdot \log^2 n)$ | One-sided |
| Section 6 | | | $O(\alpha \cdot \log n)$ | One-sided |

[†]In [51] they claim their complexity to be $O(\log \alpha \cdot \log n)$ because they only execute their PIR over the added elements, but the number of added elements is $O(\alpha)$ rather than $O(\log \alpha)$.

PSI. To do so, they propose a new oblivious data structure inspired by path ORAM, through which each party sends their "encrypted" database to the other party and can perform oblivious updates as well as compute the intersection. This approach, however, has the drawback of high storage necessity. It also does not solve the leakage of the size of the updates.

Wang et al. [51] propose a construction based on sparse PIR that leaks both the intermediate intersections and the size of the updates. Agarwal et al. [2] construct an updatable PSI scheme based on a novel dynamic structured encryption scheme, leaking the size of the update. Ling et al. [35] give a very efficient generic construction based on unbalanced PSI (where one party has a small set) and private set union, leaking the size of the updates. All these constructions only consider two-sided outputs. Moreover, [2, 35] seem to require non-trivial modifications for extended functionalities such as cardinality-PSI, because both schemes rely on using a Private Set Union (PSU) protocol after obtaining partial intersections. There is, therefore, a need for accessing the elements of the partial intersections which are basically the inputs to PSU. The knowledge of these elements already results in significant leakage for the extended functionalities. This makes [4] our closest related work, and this is why we chose to compare against them in Section 7.

A summary of our contributions with respect to the state of the art can be found in Table 1. Overall, we provide the asymptotically most efficient updatable PSI and updatable cardinality-PSI solutions that support additions and deletions freely, without leaking intermediate steps or sizes of updates.

## 4 Preliminaries

We define parties $P_1$ and $P_2$ holding sets $X^{old}, Y^{old}$, respectively, whereby $X^{old} = \{x_1, \ldots, x_{n_X}\}$ and $Y^{old} = \{y_1, \ldots, y_{n_Y}\}$ with size $|X^{old}| = n_X$ and $|Y^{old}| = n_Y$.

$(X^+, X^-)$ and $(Y^+, Y^-)$ correspond to the sets of added and removed elements for $P_1$ and $P_2$, respectively. We naturally require the updates to be valid, meaning that $X^+ \cap X^{old} = \varnothing$ and $X^- \subseteq X^{old}$, and analogously for $Y^{old}$. Finally, $X^{new} = (X^{old} \setminus X^-) \cup X^+$, and analogously for $Y^{new}$.

Furthermore, we use $[m]$ to denote the set $\{1, 2, \ldots, m\}$ of integers between 1 and $m$. We denote by $\kappa$ and $\lambda$ the computational and statistical security parameters, respectively.

*Hybrid Models.* We identify the ideal functionality for a task F with $\mathcal{F}_{\mathrm{F}}$, and with $\Pi_{\mathrm{F}}$ we denote the corresponding protocol realizing such an ideal functionality. For two-party ideal functionalities (resp. protocols) we write $(O_1; O_2) \leftarrow \mathcal{F}_{\mathrm{F}}(I_1; I_2)$ (resp. $(O_1; O_2) \leftarrow \Pi_{\mathrm{F}}(I_1; I_2)$) to indicate the execution of the ideal functionality (resp. protocol) where $I_i$ is the input and $O_i$ is the output of party $P_i$ for $i \in \{1, 2\}$. When the functionality receives a common input $CI$ agreed upon by both parties, we write $\mathcal{F}_{\mathrm{F}}(CI; I_1; I_2)$ as shorthand for $\mathcal{F}_{\mathrm{F}}((CI, I_1); (CI, I_2))$. In this case, we assume that the common input is also leaked to the adversary.

We consider protocols defined in hybrid models. Specifically, a two-party protocol $\Pi_{\mathrm{F}}$ (realizing a functionality $\mathcal{F}_{\mathrm{F}}$) is in the $\mathcal{F}_{\mathrm{G}}$-hybrid model if the protocol is run between two parties that can interact with a trusted-third party that computes the functionality $\mathcal{F}_{\mathrm{G}}$.

We recall the composition theorem [21, Theorem 7.3.3]: if a protocol $\Pi_{\mathrm{F}}$ securely realizes $\mathcal{F}_{\mathrm{F}}$ in the $\mathcal{F}_{\mathrm{G}}$-hybrid model and a protocol $\Pi_{\mathrm{G}}$ securely realizes $\mathcal{F}_{\mathrm{G}}$, then we can construct a new composed protocol $\Pi'_{\mathrm{F}}$ that executes $\Pi_{\mathrm{F}}$, and whenever $\Pi_{\mathrm{F}}$ invokes the trusted functionality $\mathcal{F}_{\mathrm{G}}$, the call is replaced by an execution of $\Pi_{\mathrm{G}}$. The composed protocol $\Pi'_{\mathrm{F}}$ securely realizes $\mathcal{F}_{\mathrm{F}}$.

*Secret Sharing.* A *t-out-of-n* secret sharing scheme [47], shares the given secret information $s$ among $n$ parties and any $t$ parties can recover it together. In this work, we consider *2-out-of-2* secret sharing schemes, as follows.

**Definition 1 (*2-out-of-2* Secret Sharing Scheme).** *Let $\kappa \in \mathbb{N}$ be a security parameter. We define a 2-out-of-2* secret sharing scheme *as the following tuple of probabilistic polynomial time (*PPT*) algorithms.*

- $\mathsf{SS.SetUp}(1^\kappa)$*: given the security parameter $\kappa$ as input, it outputs some public parameters* $\mathsf{SS.param}$*. We will assume these public parameters are input to all the other algorithms.*
- $\mathsf{SS.Share}(x)$*: given an element $x$ as input, it outputs two secret shares $s^{(1)}$ and $s^{(2)}$.*
- $\mathsf{SS.Combine}(s^{(1)}, s^{(2)})$*: given two compatible secret shares $s^{(1)}$ and $s^{(2)}$, it outputs a string* $\mathsf{res}$*.*

Secret sharing schemes have two main properties: *correctness* and *secrecy*, which are formally defined in Appendix C. Basically, correctness means when sharing and combining are done honestly, the correct secret would be recovered. Secrecy means that when less than $t$ shares are combined, they do not reveal the secret $s$. In this work we will focus on arithmetic secret shares.

**Functionality 1** Combine and Permute $\mathcal{F}_{\text{CnP}}$

*Parameters.* The secret sharing scheme parameters $\mathsf{SS.param}$.
*Inputs.* $P_1$ inputs $\{s_i^{(1)}\}_{i\in[m]}$, $P_2$ inputs $\{s_i^{(2)}\}_{i\in[m]}$ and permutation $\pi(\cdot)$ of $m$ elements.
*Outputs.* $P_1$ receives $\pi(\{\mathsf{SS.Combine}(s_i^{(1)}, s_i^{(2)})\}_{i\in[m]})$, $P_2$ receives nothing. The adversary receives $m$.

**Functionality 2** One-Sided Private Set Intersection $\mathcal{F}_{\text{PSI}}$

*Inputs.* $P_1$ inputs $X$, $P_2$ inputs $Y$.
*Outputs.* $P_1$ receives intersection $I = X \cap Y$, $P_2$ receives nothing. The adversary receives $|X|, |Y|$.

*Combine and Permute.* The Combine and Permute (CnP) [14, 49] protocol is a cryptographic primitive that ensures the privacy of shuffling within secret sharing schemes. The main idea is that when shares are being reconstructed, there exists a link between the share and the reconstructed value, which, in larger protocols with shares being obtained from different subroutines, may leak unwanted information. CnP protocols can be used in reconstruction to obtain the shared values in a random order, thus breaking the link to the previously owned shares, as done in [11]. More concretely, $P_1$ (who will receive the combined shares) inputs a set of $m$ secret shares $\{s_i^{(1)}\}_{i\in[m]}$ and $P_2$ inputs a set of $m$ secret shares $\{s_i^{(2)}\}_{i\in[m]}$ as well as a permutation of $m$ elements $\pi(\cdot)$. The protocol outputs the reconstructed shares shuffled under permutation $\pi$, $\pi(\{\mathsf{SS.Combine}(s_i^{(1)}, s_i^{(2)})\}_{i\in[m]})$ to $P_1$. The ideal functionality is given in Functionality 1.

*Circuit PSI.* Circuit PSI (cPSI) protocols are 2PC protocols that allow two parties to obtain secret shares of the intersection of their respective private sets without revealing any further information. These secret shares can be further used to compute any arbitrary circuit over the intersection by using some generic 2PC protocol. As such, depending on the specific generic 2PC protocol used afterwards, the shares can be set differently. For example, some protocols have secret shares of the value of elements, while others have secret shares of 1 with one of the parties having a correspondence to their dataset. In more details, when $P_1$ inputs set $X$ of size $n_X$ and $P_2$ inputs set $Y$, then the protocol outputs a set of $n_X$ secret shares $\{s_i^{(1)}\}_{i\in[n_x]}$ to $P_1$ and a set of $n_X$ secret shares $\{s_i^{(2)}\}_{i\in[n_x]}$ to $P_2$. While the ideal functionality for PSI is given in Functionality 2, Functionality 3 represents Circuit PSI, where we have stated both types of shares that will be used by our constructions.

*Unbalanced PSI.* Unbalanced PSI [23,36,48] refers to private set intersection protocols tailored for scenarios where one party holds a significantly smaller dataset

**Functionality 3** Circuit Private Set Intersection $\mathcal{F}_{\text{cPSI}}$

*Parameters.* The secret sharing scheme parameters $\mathsf{SS.param}$,
*Inputs.* $P_1$ inputs $X$ and $P_2$ inputs $Y$.
*Common Inputs.* Two flags, one string $\mathsf{mode} \in \{\mathsf{CA}, \mathsf{PSI}\}$, and an integer $\mathsf{side} \in \{1, 2\}$.
*Outputs.* $P_1$ and $P_2$ receive a set of $n_X$ secret shares $\{s_i^{(1)}, s_i^{(2)}\}_{i \in [n_X]}$ where

$$\mathsf{SS.Combine}\left(s_i^{(1)}, s_i^{(2)}\right) = \begin{cases} 1 & \text{if } \mathsf{mode} = \mathsf{CA} \text{ and } z_i \in X \cap Y, \\ z_i & \text{if } \mathsf{mode} = \mathsf{PSI} \text{ and } z_i \in X \cap Y, \\ 0 & \text{otherwise.} \end{cases}$$

Where the functionality parses set $Z = \{z_1, \ldots, z_{|Z|}\}$ and the set is defined as:

$$Z := \begin{cases} X & \text{if } \mathsf{side} = 1 \\ Y & \text{if } \mathsf{side} = 2 \end{cases}$$

The adversary receives $|X|, |Y|$.

**Functionality 4** One-sided Updatable Private Set Intersection $\mathcal{F}_{\text{uPSI}}$

*Inputs.* $P_1$ inputs $X^{old}, X^+, X^-$ with $|X^+|, |X^-| \leq \alpha$ where $X^+, X^-$ are valid updates for $X^{old}$, and $I^{old} = X^{old} \cap Y^{old}$, $P_2$ inputs $Y^{old}, Y^+, Y^-$ with $|Y^+|, |Y^-| \leq \alpha$ where $Y^+, Y^-$ are valid updates for $Y^{old}$.
*Outputs.* $P_1$ receives the new intersection $I^{new} = X^{new} \cap Y^{new}$, $P_2$ receives nothing. The adversary receives $|X^{old}|, |Y^{old}|, |X^{new}|, |Y^{new}|$.

than the other. These protocols optimize for this asymmetry: the communication complexity depends only on the size of the smaller set, and the computational cost for each party is proportional to the size of its own input.

## 5 Updatable PSI

### 5.1 Definition

Functionality 4 defines the one-sided updatable PSI ideal functionality. The two parties input their previous sets $X^{old}, Y^{old}$ as well as the sets of added and removed elements $X^+, X^-, Y^+, Y^-$. $P_1$ also inputs the previously computed intersection $I^{old} = X^{old} \cap Y^{old}$. With these inputs, $\mathcal{F}_{\text{upsi}}$ outputs the new intersection $I^{new} = X^{new} \cap Y^{new}$ to $P_1$ only, where $X^{new} := (X^{old} \setminus X^-) \cup X^+$ and analogously for $Y^{new}$. The leakage only includes the sizes of old and new sets (since new sets will be used as old sets in the next execution between the same parties). Note that since we are in the honest-but-curious setting, by chaining instantiations of this functionality we obtain a functionality for an arbitrary amount of updates. For a more detailed explanation, we refer to Appendix A.

### 5.2 Construction of Disjoint Sets

Let $P_1$ own $X^{old}, X^+, X^-$ and $P_2$ own $Y^{old}, Y^+, Y^-$. In general, we assume $|X^+|, |X^-| \ll |X^{old}|$ (and analogously for $Y$). The goal is to compute intersection $I^{new} = X^{new} \cap Y^{new}$ of $X^{new} := (X^{old} \backslash X^-) \cup X^+$ and $Y^{new} := (Y^{old} \backslash Y^-) \cup Y^+$ leveraging the knowledge of the old intersection $I^{old} = X^{old} \cap Y^{old}$. For the sake of efficiency, instead of computing $I^{new}$ from $X^{new}$ and $Y^{new}$, we compute a constant number of intermediate intersections between one small set and one large set.

The main candidates for small sets are the four update sets $X^+, X^-, Y^+, Y^-$. The first intuition is to use each update set, individually. The added elements $X^+ \cap Y^{new}$ and $Y^+ \cap X^{new}$ as well as the deleted elements $X^- \cap Y^{old}$ and $Y^- \cap X^{old}$, and then compute $I^{new}$ as:

$$(I^{old} \backslash ((X^- \cap Y^{old}) \cup (Y^- \cap X^{old}))) \cup (X^+ \cap Y^{new}) \cup (Y^+ \cap X^{new}). \quad (1)$$

However, we observe that there are elements appearing in more than one of these intermediate intersections. For example $X^+ \cap Y^+$ appears in both $X^+ \cap Y^{new}$ and $Y^+ \cap X^{new}$. There could be some issues with this solution. Firstly, if the union of the intermediate intersection is not performed obliviously, this setup clearly leaks the elements in $X^+ \cap Y^+$ and $X^- \cap Y^-$, which is an unauthorized leakage. Secondly, computing the cardinality by simply summing the secret-shared values would not work, because elements that appear more than once would be counted multiple times.

It is therefore more practical to compute four mutually disjoint intersections. To achieve this, we isolate the elements in $X^+ \cap Y^+$ and $X^- \cap Y^-$, ensuring that each element is included in exactly one of the four intersections. In Equation (1), since $X^+ \cap Y^+$ is both in $X^+ \cap Y^{new}$ and $Y^+ \cap X^{new}$, we can replace $X^{new}$ by $X^{new} \backslash X^+$ which corresponds to $Y^+ \cap X^{old} \backslash X^-$, and similarly since $X^- \cap Y^-$ is both in $Y^- \cap X^{old}$ and $X^- \cap Y^{old}$, we replace $Y^{old}$ with $Y^{old} \backslash Y^-$ and obtain the intersection $X^- \cap Y^{old} \backslash Y^-$. Putting it all together we obtain:

$$\begin{aligned} I^{new} = \big(I^{old} \backslash \big((X^- \cap Y^{old}) \uplus (Y^- \cap (X^{old} \backslash X^-))\big)\big) \\ \uplus \big((X^+ \cap Y^{new}) \uplus (Y^+ \cap (X^{old} \backslash X^-)\big) \end{aligned}$$

where $\uplus$ denotes the disjoint union of sets, namely, $A \uplus B = A \cup B$ if $A \cap B = \emptyset$, otherwise it defines their co-product. Appendix B provides a numerical example[4].

**Lemma 1.** *Let $X^{old}, X^+, X^-, Y^{old}, Y^+, Y^-$ be sets and $X^+, X^-$ (respectively $Y^+, Y^-$) are valid updates for $X^{old}$ (respectively $Y^{old}$) and let $I^{old} = X^{old} \cap Y^{old}$. Set*

$$I_1 := (X^{new} \cap Y^+) \qquad I_2 := X^+ \cap (Y^{old} \backslash Y^-)$$
$$I_3 := (X^{old} \cap Y^-) \qquad I_4 := (X^- \cap (Y^{old} \backslash Y^-))$$

[4] Note that this way of assembling the sets is only one of four possible ways we can achieve through this method (for both added and deleted elements, one may choose which party gets the larger set), which are, under most circumstances, interchangeable.

*Then*

$$X^{new} \cap Y^{new} = (I^{old} \setminus (I_3 \cup I_4)) \cup (I_1 \cup I_2) \tag{2}$$

$$I_1 \cap I_2 = I_3 \cap I_4 = I^{old} \cap I_1 = I^{old} \cap I_2 = \emptyset \tag{3}$$

*Moreover,* $I^+ = I_1 \cup I_2, I^- = I_3 \cup I_4$ *are valid updates for* $I^{old}$*.*

We refer to Appendix C for a formal proof of the lemma.

**Corollary 1.** *Let* $X^{old}, X^+, X^-, Y^{old}, Y^+, Y^-, I^{old}$, $I_1, I_2, I_3, I_4$ *be sets defined as in Lemma 1. Then*

$$|X^{new} \cap Y^{new}| = |I^{old}| - (|I_3| + |I_4|) + (|I_1| + |I_2|).$$

### 5.3 Protocol Description

To compute the updated intersection while improving the performance with respect to the naive PSI solution, our protocol calls a protocol securely realizing the ideal functionality $\mathcal{F}_{cPSI}$ (Functionality 3) with the $\mathsf{PSI}$ mode four disjoint times with the sets $X^+, X^-, Y^+, Y^-$. Since the size of the updates are much smaller than the sizes of $X$ and $Y$, performing four calls to an unbalanced PSI is more efficient than computing the intersection over the updated sets from scratch.

To address the leakage of the size of the updates, the first step of the protocol is for each party to locally add some dummy elements to these sets to reach a common threshold size set of $\alpha$. Thus, leaking the size of the updates is no longer an issue, since it is known beforehand. We also discuss how to pick $\alpha$ later on in this section.

Secondly, to address the leakage regarding the reconstructed output from the cPSI executions, we take two different measures. First, when executing the four instances of cPSI, we make sure to define the big sets in such a way that the four resulting intersections are disjoint, to avoid the leakage, as explained in Section 5.2. The second measure is when reconstructing the shares: To avoid the party obtaining the intersections (and thus reconstructing the shares) to know which element came from which intersection, we reconstruct them by calling a protocol that securely realizes the ideal functionality $\mathcal{F}_{CnP}$ twice: one for added elements $(X^+, Y^+)$ and one for removed elements $(X^-, Y^-)$. Therefore, the reconstruction will not disclose the specific outputs of each intersection. Finally, the party obtaining the new added and deleted elements can update the old intersection. The description of this protocol is in Protocol 1.

**On the Selection of $\alpha$.** For practical scenarios, we note that there is a trade-off in the choice of the parameter $\alpha$. If $\alpha$ is small compared to the number of updates our protocol will propose a small overhead (linear in $\alpha$) while a larger $\alpha$ will converge to regular cPSI. In particular, in our empirical evaluation (see section 7), we chose $\alpha = O(log(n))$.

**Protocol 1** One-sided Updatable Private Set Intersection $\Pi_{\text{uPSI}}$

*Parameters:* Threshold set size $\alpha$.
*Inputs:* $P_1$ inputs $X^{old}$, $X^+$, $X^-$ with $|X^+|, |X^-| \leq \alpha$, and, $I^{old} = X^{old} \cap Y^{old}$, $P_2$ inputs $Y^{old}$, $Y^+$, $Y^-$ with $|Y^+|, |Y^-| \leq \alpha$.
*Outputs:* $P_1$ outputs $I^{new} = X^{new} \cap Y^{new}$. $P_2$ outputs nothing.

1. $P_1$ and $P_2$ check locally the sets $X^+, X^-$ and $Y^+, Y^-$ valid updates for $X^{old}$ and $Y^{old}$ and add dummy elements until $|X^+| = |X^-| = |Y^+| = |Y^-| = \alpha$, as well as $|X^{old} \backslash X^-| = |X^{old}|$ and $|Y^{old} \backslash Y^-| = |Y^{old}|$.
2. $P_1$ and $P_2$ execute four circuit-PSIs as follows:
   - $(\boldsymbol{s}^{(1)}_{Y^+}; \boldsymbol{s}^{(2)}_{Y^+}) \leftarrow \mathcal{F}_{\text{cPSI}}(\mathsf{PSI}, 2;\ X^{new}; Y^+)$.
   - $(\boldsymbol{s}^{(1)}_{X^+}; \boldsymbol{s}^{(2)}_{X^+}) \leftarrow \mathcal{F}_{\text{cPSI}}(\mathsf{PSI}, 1;\ X^+; Y^{old} \backslash Y^-)$.
   - $(\boldsymbol{s}^{(1)}_{Y^-}; \boldsymbol{s}^{(2)}_{Y^-}) \leftarrow \mathcal{F}_{\text{cPSI}}(\mathsf{PSI}, 2;\ X^{old}; Y^-)$.
   - $(\boldsymbol{s}^{(1)}_{X^-}; \boldsymbol{s}^{(2)}_{X^-}) \leftarrow \mathcal{F}_{\text{cPSI}}(\mathsf{PSI}, 1;\ X^-; Y^{old} \backslash Y^-)$.
3. $P_1$ and $P_2$ run the following two CnPs where $||$ denotes concatenation:
   - $(I^-; \perp) \leftarrow \mathcal{F}_{\text{CnP}}(\boldsymbol{s}^{(1)}_{Y^-} \| \boldsymbol{s}^{(1)}_{X^-}; \boldsymbol{s}^{(2)}_{Y^-} \| \boldsymbol{s}^{(2)}_{X^-})$.
   - $(I^+; \perp) \leftarrow \mathcal{F}_{\text{CnP}}(\boldsymbol{s}^{(1)}_{Y^+} \| \boldsymbol{s}^{(1)}_{X^+}; \boldsymbol{s}^{(2)}_{Y^+} \| \boldsymbol{s}^{(2)}_{X^+})$.
4. $P_1$ outputs $I^{new} = (I^{old} \backslash I^-) \cup I^+$.

Fig. 2: Our generic protocol for standard updatable private set intersection.

**Construction of Dummy Elements.** In order to avoid potential collisions in dummy elements added by both parties, prefixes specific to each party can be utilized. As a concrete example, the $j$-th dummy element of $P_i$ can be the concatenation of the identifier of $P_i$, the index of the dummy element (e.g. $j$), and a specific fixed string such that is not in the domain of items of the set.

### 5.4 Complexity Analysis

For ease of notation, let us assume $n = |X^{old}| = |Y^{old}|$ (and therefore $|X^{new}|$, $|Y^{new}| \leq n + \alpha$). The complexity of our updatable PSI protocol depends on the complexity of the underlying building blocks, namely the protocols $\Pi_{\text{cPSI}}$ and $\Pi_{\text{CnP}}$ instantiating their respective ideal functionalities. Indeed, the communication cost is computed as 4 times the communication cost of $\Pi_{\text{cPSI}}$ in step 2, and 2 times the cost of $\Pi_{\text{CnP}}$ in step 3. Considering an asymmetric circuit PSI like [48] with a communication complexity of $O(\alpha \cdot \log(n/\alpha))$ and a CnP like [13] with a communication complexity of $O(\alpha \cdot \log \alpha)$, we get an asymptotic, communication complexity of $O(\alpha \cdot \log n/\alpha + \alpha \cdot \log \alpha)$. Analogously, for computation time we will have the 4 parallel independent executions of $\Pi_{\text{cPSI}}$ in step 1 followed by the 2 parallel executions of $\Pi_{\text{CnP}}$ and finished (for $P_1$) by the reconstruction of $I^{new}$. Using, for example, the cPSI from [23] leveraging the linear complexity in the update sets and sublinear in the big sets, we could get an asymptotic computation complexity of $O(\alpha \cdot \log n + \alpha \cdot \log \alpha)$.

**Functionality 5** One-sided Updatable Cardinality-Private Set Intersection $\mathcal{F}_{\text{uPSI-card}}$

*Inputs.* $P_1$ inputs $X^{old}, X^+, X^-$ with $|X^+|, |X^-| \leq \alpha$ and $c^{old} = |X^{old} \cap Y^{old}|$, $P_2$ inputs $Y^{old}, Y^+, Y^-$ with $|Y^+|, |Y^-| \leq \alpha$.
*Outputs.* $P_1$ receives the cardinal of the new intersection $c^{new} = |X^{new} \cap Y^{new}|$, $P_2$ receives nothing.
The adversary receives $|X^{old}|, |Y^{old}|, |X^{new}|, |Y^{new}|$.

### 5.5 Correctness and Security

**Theorem 1.** *Protocol $\Pi_{uPSI}$ presented in Figure 1 securely realizes the ideal functionality $\mathcal{F}_{uPSI}$ in the $(\mathcal{F}_{cPSI}, \mathcal{F}_{CnP})$-hybrid model against a semi-honest adversary, assuming the secret sharing scheme holds secrecy.*

*Proof Sketch.* For the complete proof we refer to Appendix C, here we provide a sketch. Lemma 1 ensures the correctness of the protocol, since the $j$-th execution of $\mathcal{F}_{\text{cPSI}}$ in protocol $\Pi_{\text{uPSI}}$ computes the secret shares of the elements in set $I_j$ defined in the lemma. Regarding privacy, notice that the parties in the protocol only interact with the ideal functionalities. In particular, when invoking $\mathcal{F}_{\text{cPSI}}$, the parties receive secret shares as output. By the secrecy of the secret-sharing scheme, no information about the underlying sets is revealed beyond their cardinalities, which the parties fix to $\alpha$ using dummy inputs. Therefore, even the cardinalities do not leak.

When invoking $\mathcal{F}_{\text{CnP}}$, party $P_1$ obtains sets $I^-$ and $I^+$. It is important to note that, thanks to Equation (3) in Lemma 1, the outputs of these functionalities can indeed be interpreted as sets, since there are no repeated elements. We also note that sets $I^-$ and $I^+$ can be efficiently derived from $I^{old}$ and the output of the ideal functionality $\mathcal{F}_{\text{uPSI}}$ realized by the protocol, since, by Lemma 1, they constitute valid updates for $I^{old}$. Thus, the outputs of $\mathcal{F}_{\text{CnP}}$ can be simulated using the input and output of $P_1$ in the ideal world, only.

## 6 Updatable Cardinality-PSI

We present our new protocol for one-sided updatable cardinality-PSI and extend it to support the payload sum feature as well. Our protocol uses the cPSI functionality in $\mathsf{CA}$ mode and arithmetic secret sharing to address the two leakages identified in Section 2.

### 6.1 Definition

Functionality 5 defines the one-sided updatable cardinality-PSI ideal functionality. The two parties input their previous sets $X^{old}, Y^{old}$ as well as the sets of added and removed elements $X^+, X^-, Y^+, Y^-$, and $P_1$ also inputs the previously computed cardinality result $c^{old} = |X^{old} \cap Y^{old}|$. With these inputs, $\mathcal{F}_{\text{uPSI-card}}$ (Functionality 5) outputs the cardinality of the new intersection $c^{new} = |X^{new} \cap$

**Protocol 2** One-sided Updatable Cardinality-Private Set Intersection $\Pi_{\text{uPSI-card}}$

*Parameters:* Threshold set size $\alpha$.
*Inputs:* $P_1$ inputs $X^{old}, X^+, X^-$, as well as $c^{old} = |X^{old} \cap Y^{old}|$.
$P_2$ inputs $Y^{old}, Y^+, Y^-$.
*Outputs:* $P_1$ outputs $c^{new} = |X^{new} \cap Y^{new}|$. $P_2$ outputs nothing.

1. $P_1$ and $P_2$ check locally the sets $X^+, X^-$ and $Y^+, Y^-$ valid updates for $X^{old}$ and $Y^{old}$ and add dummy elements until $|X^+| = |X^-| = |Y^+| = |Y^-| = \alpha$, as well as $|X^{old} \backslash X^-| = |X^{old}|$ and $|Y^{old} \backslash Y^-| = |Y^{old}|$.
2. $P_1$ and $P_2$ four circuit-PSIs for cardinality as follows:
   - $\{(\boldsymbol{c}^{(1)}_{Y^+,i}; \boldsymbol{c}^{(2)}_{Y^+,i})\}_{i\in[\alpha]} \leftarrow \mathcal{F}_{\text{cPSI}}(\mathsf{CA}, 2;\ X^{new}; Y^+)$.
   - $\{(\boldsymbol{c}^{(1)}_{X^+,i}; \boldsymbol{c}^{(2)}_{X^+,i})\}_{i\in[\alpha]} \leftarrow \mathcal{F}_{\text{cPSI}}(\mathsf{CA}, 1;\ X^+; Y^{old} \backslash Y^-)$.
   - $\{(\boldsymbol{c}^{(1)}_{Y^-,i}; \boldsymbol{c}^{(2)}_{Y^-,i})\}_{i\in[\alpha]} \leftarrow \mathcal{F}_{\text{cPSI}}(\mathsf{CA}, 2;\ X^{old}; Y^-)$.
   - $\{(\boldsymbol{c}^{(1)}_{X^-,i}; \boldsymbol{c}^{(2)}_{X^-,i})\}_{i\in[\alpha]} \leftarrow \mathcal{F}_{\text{cPSI}}(\mathsf{CA}, 1;\ X^-; Y^{old} \backslash Y^-)$.
3. $P_j$, $j \in \{1, 2\}$, locally computes

$$c^{(j)} = \sum_{i\in[\alpha]} \boldsymbol{c}^{(j)}_{Y^+,i} + \boldsymbol{c}^{(j)}_{X^+,i} - \boldsymbol{c}^{(j)}_{Y^-,i} - \boldsymbol{c}^{(j)}_{X^-,i}.$$

   $P_2$ sends $c^{(2)}$ to $P_1$.
4. $P_1$ outputs $c^{new} = c^{old} + \mathsf{SS.Combine}(c^{(1)}, c^{(2)})$.

Fig. 3: Our generic protocol for updatable cardinality-private set intersection.

$Y^{new}|$ to $P_1$ only, where $X^{new} := (X^{old} \setminus X^-) \cup X^+$ and analogously for $Y^{new}$. Similarly to the previous section, by chaining instantiations of this functionality we obtain a functionality for an arbitrary amount of updates.

### 6.2 Protocol Description

Similar to the construction of $\Pi_{\text{uPSI}}$ in Protocol 1, the protocol $\Pi_{\text{uPSI-card}}$ for cardinality in Protocol 2 executes four circuit-PSIs in $\mathsf{CA}$ mode, using the sets $I_1, \ldots, I_4$ defined in Lemma 1. Namely, the outputs of the cPSI sub-protocols are shares of binary vectors, where a 0 in the $j$-th position indicates that the $j$-th element is not in the intersection, and a 1 indicates that it is. Moreover, we assume a linear secret-sharing scheme (alternatively, additively homomorphic encryption can be employed, as in [31]).

Analogously to the construction in the previous section, to address the leakage of the size of the updates, each party locally adds dummy elements to the update sets to reach a common threshold set size of $\alpha$. By using the $\mathsf{CA}$ mode of cPSI and the linearity of arithmetic secret shares, each party can locally add all the shares received from the added intersections and subtract all the shares

received from the deleted intersections to obtain a secret share of the change in cardinal from the old to the new intersection. Then, this value can be reconstructed by $P_1$ and added to the old cardinal. Note that in the cardinality case, the fact that the four intersections computed are disjoint is a requirement to guarantee the correctness of the protocol. Moreover, since the shares of individual elements in the intersection are not reconstructed one by one, no leakage occurs about their position within the sets $I_1, \ldots, I_4$. Therefore, the CnP functionality is no longer required.

### 6.3 Complexity Analysis

For ease of notation, let us assume $n = |X^{old}| = |Y^{old}|$ (and therefore $|X^{new}|$, $|Y^{new}| \leq n+\alpha$). The communication cost and the computation cost is computed as 4 times that of $\Pi_{\text{cPSI}}$. The remaining operations consists of simple sum and reconstruction of one single secret share. Hence, as in the previous case, considering an asymmetric circuit PSI like [48], we get an asymptotic complexity for communication cost of $O(\alpha \cdot \log(n/\alpha))$, and considering the proposal in [23], we get a complexity of $O(\alpha \cdot \log n)$ for computation cost.

### 6.4 Security

**Theorem 2.** *Protocol $\Pi_{uPSI-card}$ presented in Figure 2 securely and correctly realizes the ideal functionality $\mathcal{F}_{uPSI-card}$ described as Functionality 5 in the $\mathcal{F}_{cPSI}$-hybrid model against a semi-honest adversary, assuming the arithmetic secret sharing holds secrecy and correctness.*

*Proof Sketch.* Correctness follows directly from Corollary 1 and from the linearity of the secret sharing scheme.[5] The proof of privacy is almost trivial; nevertheless, we describe the simulator. The simulator samples one secret share, for example $\mathbf{c}^{(1)}_{X^+,1}, \mathbf{c}^{(2)}_{X^+,1}$, to form a sharing of the value $\delta = c^{\text{new}} - c^{\text{old}}$, while sampling the remaining shares as sharings of 0. It then simulates the interactions with $\mathcal{F}_{\text{cPSI}}$ by setting the outputs of these interactions to the sampled shares. The proof follows by a standard hybrid argument relying on the secrecy of secret-shared values.

### 6.5 Updatable Payload-Sum-PSI

Private set intersection for payload sum is a variant of PSI, where each element in the sets has a related payload $v_x$ and the objective is to output the sum of the payloads of the elements in the intersection, i.e., $\sum_{x \in X^{old} \cap Y^{old}} v_x$.

To modify $\Pi_{\text{uPSI-card}}$ into $\Pi_{\text{uPSI-sum}}$, we only need to slightly change the underlying base circuit-PSI protocol in PSI mode. This cPSI now will output

[5] As a caveat, linear secret-sharing schemes are usually defined over finite fields (or finite rings). Strictly speaking, this means our protocol computes the cardinality modulo the size of the underlying field. The common way to address this is simply to choose a field large enough to avoid wrap-around.

secret shares of the value $v_x$ rather than the actual element. Then, all the steps of the protocol described in Figure 2 are executed identically and $P_1$ obtains the resulting payload sum.

Regarding the more generic functionality of circuit-PSI, designing an efficient updatable version from this architecture comes with impractical challenges. For a more detailed study of these, as well as some solutions to partially circumvent them we refer to Appendix D.

## 7 Performance Evaluation

### 7.1 Experimental Setting

We have implemented[6] our constructions for updatable PSI and updatable cardinality-PSI in Rust. Regarding the circuit-PSI protocol $\Pi_{\text{cPSI}}$, we use the proposal by Karakoç and Küpçü [31], since, on top of being very efficient due to its leveraging of oblivious programmable pseudo-random function and Cuckoo tables, it also offers flexibility for its output. Indeed, the output can either be secret shares of 1 or 0 and suit the computation of the cardinality, or secret shares of the value of the element or 0 and suit standard uPSI. Some additional optimizations to our protocol $\Pi_{\text{uPSI-card}}$ were made thanks to this use and are explained in Appendix E. To implement this scheme we have chosen to use the secure equality test by Ciampi and Orlandi in [17] with the `oblivious_transfer_protocols` library [24] for Oblivious Transfer extension and the `aes` library [46], as well as the Paillier encryption scheme [39] with the `kzen-paillier` library [7] for the homomorphic encryption scheme. The underlying additive arithmetic secret sharing scheme uses a simple addition in $\mathbb{Z}_L$ with $L$ being an upper bound to the domain of the secret sharing. Finally, for the combine and permute protocol $\Pi_{\text{CnP}}$, we use the naïve approach described in [13, Appendix C] where $P_1$ encrypts its shares with an additively homomorphic encryption scheme, sends them to $P_2$, who reconstructs the output while being encrypted, shuffles them, and finally sends them back to $P_1$ for decryption.

We believe that the most relevant work to compare against our solution is [4][7], since it is the only solution that implements cardinality-PSI, payload-sum-PSI, and the standard PSI when sets are updated. Nevertheless, in that solution, $P_2$ constructs an oblivious data structure of $Y$ and preliminarily sends it to $P_1$, who further executes all queries locally. Hence, compared to our solution, the one in [4] requires a much larger amount of storage and a significant communication overhead before the query execution starts.

All benchmarks were obtained on an Intel Core i7-7800X (6C/12T, 3.5 GHz, AVX-512), with 128 GB of RAM running Ubuntu 20.04 operating system. The parties are executed in a different thread each, and communicate through localhost. As in the previous work [4], we use the Linux `tc` command to simulate different network settings, with LAN at 0.2ms network latency and 1Gbps

[6] Implementation available at `https://gitlab.eurecom.fr/project-spring-2025/PSI/`

[7] Implementation available: `https://github.com/ruidazeng/upsi-revisited`

bandwidth, and WAN at 80ms network latency and testing several bandwidths of 200Mbps, 50Mbps, and 5Mbps.

The size of the datasets $n$ ranges from $2^{16}$ to $2^{17}$. We had to stop at this size because the solution in [4] could not be executed with larger sizes (the authors also mentioned that they had to increase their RAM for larger datasets). For larger datasets, we only tested our own protocols. The size of updates $\alpha$ varies from $2^4$ to $2^6$. Note that the sizes of the new (updated) sets will be bounded by $n \pm \alpha$.

### 7.2 Performance Results

We evaluate our protocols $\Pi_{\text{uPSI}}$ and $\Pi_{\text{card-uPSI}}$ in terms of communication and computation cost, and compare them with the performance of protocols proposed by Badrinarayan et al. [4] supporting both addition and deletion (i.e., Section 4, in [4]). Table 2 depicts the obtained measurements. We also evaluate the runtime and communication cost for larger sets frow which we were not able to run the code from [4] and these results are shown in Table 3.

**Communication Cost.** As expected, our solution for updatable cardinality-PSI is slightly more efficient than our standard updatable PSI due to the additional use of the combine and permute protocol in our updatable PSI. Compared to [4], in general, our solution incurs higher communication cost when the updates are smaller. The communication cost of our solutions grows linearly with the updates more slowly than the one in [4]. Therefore, we outperform [4] as the update size grows.

It is worth noting that our solutions and [4] follow very different approaches leading to very different architectures. [4] uses an oblivious data structure for each dataset that is stored by the other party and can be obliviously updated and queried, whereas our solutions compute these updates with each party only having access to its (updated) dataset as well as the old intersection (or cardinal thereof). Therefore, [4] inherently requires a much larger local storage cost, while sending less information, in general. Furthermore, these oblivious data structures must be sent to the other party beforehand, adding communication cost.

With respect to the communication cost in larger sets ($n = 2^{20}$), we see that it becomes notably large. This is mostly due to the specific OT extension library in Rust that is used in our implementation, which is not as highly optimized as some other C libraries, and to not implementing the Silent OT extension [9].

**Computation Time.** Similar to the communication cost, as expected, the computation time is lower for the cardinality case than the standard PSI case. As for comparison results, our protocol outperforms [4], by $11\times$ to $40\times$ for updatable PSI and by $14\times$ to $107\times$ for updatable cardinality-PSI in the LAN setting, and almost all of the WAN settings. It is worth noting that our implementation seems to be more sensitive to bandwidth because, due to the design choice, our code incurs more latency than in the performance results of the implementation from [4]. Indeed, in our benchmarks, the running time increases more significantly when the bandwidth is smaller (for example, $5Mbps$). Future work can be done to mitigate this through code optimizations.

Table 2: Comparison of our constructions with state-of-the-art constructions from [4]. Best performances are highlighted in **bold**, and when our protocol outperforms [4], the corresponding cells are emphasized in green, otherwise, i.e., when [4] performs better, the actual cells are highlighted in blue.

| | $n$ | $\alpha$ | Protocol | Comm. (MB) | Online Run Time (s) | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | LAN | 200 Mbps | 50 Mbps | 5 Mbps |
| PSI | $2^{16}$ | $2^4$ | [4] | **54.11** | 113.23 | 117.01 | 182.95 | 183.06 |
| | | $2^5$ | | 107.11 | 218.05 | 229.42 | 368.42 | 367.27 |
| | | $2^6$ | | 211.72 | 424.17 | 447.79 | 735.09 | 738.92 |
| | | $2^4$ | Protocol 1 | 96.42 | **6.00** | **11.99** | **22.22** | **159.16** |
| | | $2^5$ | | **97.47** | **7.44** | **13.62** | **23.24** | **162.28** |
| | | $2^6$ | | **99.6** | **10.46** | **16.29** | **27.38** | **167.58** |
| | $2^{17}$ | $2^4$ | [4] | **55.16** | 114.89 | 119.29 | 185.95 | **187.62** |
| | | $2^5$ | | **108.90** | 222.46 | 232.35 | 374.99 | **375.51** |
| | | $2^6$ | | 217.65 | 438.58 | 462.03 | 762.45 | 764.81 |
| | | $2^4$ | Protocol 1 | 191.69 | **9.64** | **18.56** | **42.17** | 311.89 |
| | | $2^5$ | | 192.74 | **11.44** | **19.8** | **40.63** | 314.37 |
| | | $2^6$ | | **194.88** | **14.2** | **22.81** | **43.76** | **321.26** |
| Cardinality PSI | $2^{16}$ | $2^4$ | [4] | **53.47** | 108.65 | 114.04 | 178.37 | 179.77 |
| | | $2^5$ | | 106.20 | 211.80 | 223.37 | 360.40 | 362.58 |
| | | $2^6$ | | 210.48 | 416.50 | 438.17 | 722.02 | 728.79 |
| | | $2^4$ | Protocol 2 | 91.55 | **3.88** | **8.89** | **18.95** | **150.06** |
| | | $2^5$ | | **94.58** | **3.97** | **9.23** | **19.46** | **154.6** |
| | | $2^6$ | | **97.37** | **3.86** | **9.73** | **19.81** | **158.42** |
| | $2^{17}$ | $2^4$ | [4] | **54.92** | 112.32 | 116.89 | 183.10 | **182.38** |
| | | $2^5$ | | **109.07** | 217.40 | 229.49 | 369.83 | 372.83 |
| | | $2^6$ | | 216.69 | 429.72 | 451.35 | 749.89 | 751.44 |
| | | $2^4$ | Protocol 2 | 182.17 | **7.65** | **16.43** | **35.5** | 294.69 |
| | | $2^5$ | | 187.39 | **7.88** | **15.29** | **36.68** | **302.56** |
| | | $2^6$ | | **191.33** | **7.87** | **17.31** | **37.42** | **309.43** |

Regarding computation time in the scenario with the largest sets, we observe that our protocol with $n = 2^{20}$ elements even outperforms in the LAN setting than of [4] with $n = 2^{16}$ elements. The performance drops in the WAN settings, which is explainable because of the combination of the sensitivity of our implementation to bandwidth and the high communication cost.

### 7.3 Evaluation on Real Data

To empirically evaluate our constructions in a real-life use case, we consider a scenario of spam detection. In such a scenario, an entity puts into place a spam filter by setting up a blacklist (for example in relation to the email addresses), and computes a PSI with its clients' received emails (their senders) to remove the spam. To analyze our constructions' efficiency in this situation, we have opted to use real data from the Enron mail database.[8] We randomly extracted

[8] `https://www.kaggle.com/datasets/wcukierski/enron-email-dataset/code`.

Table 3: Run-time and communication cost for our constructions with big sets.

| $n$ | $\alpha$ | Protocol | Comm. (MB) | Online Run Time (s) | | | |
|---|---|---|---|---|---|---|---|
| | | | | LAN | 200 Mbps | 50 Mbps | 5 Mbps |
| $2^{20}$ | $2^4$ | Protocol 1 | 1525.53 | 96.52 | 150.50 | 319.10 | 2489.80 |
| | $2^5$ | | 1526.5 | 98.67 | 140.44 | 307.39 | 2492.97 |
| | $2^6$ | | 1528.72 | 100.66 | 157.01 | 309.14 | 2495.41 |
| | $2^4$ | Protocol 2 | 1451.6 | 90.54 | 140.11 | 305.08 | 2370.96 |
| | $2^5$ | | 1487 | 93.74 | 136.93 | 295.56 | 2432.86 |
| | $2^6$ | | 1506.98 | 96.32 | 147.65 | 302.41 | 2458.83 |

Table 4: Benchmarking of our constructions with real-life data from the ENRON mail database.

| $\alpha$ | Protocol | Comm. (MB) | Offline Runtime (s) | Online Runtime (s) | | | |
|---|---|---|---|---|---|---|---|
| | | | | LAN | 200 Mbps | 50 Mbps | 5 Mbps |
| $2^6$ | Protocol 1 | 380.85 | 9.29 | 20.51 | 35.4 | 77.92 | 616.73 |
| $2^8$ | | 393.96 | 14.1 | 39.43 | 53.96 | 98.39 | 656.47 |
| $2^{10}$ | | 443.1 | 33.45 | 108.6 | 126.13 | 170.59 | 796.25 |
| $2^6$ | Protocol 2 | 374.23 | 9.29 | 14.27 | 26.93 | 70.12 | 601.85 |
| $2^8$ | | 387.85 | 14.1 | 14.31 | 32.12 | 71.52 | 624.39 |
| $2^{10}$ | | 426.48 | 33.45 | 14.67 | 32.35 | 76.97 | 686.11 |

two separate datasets of the same size (around $2^{18}$) from the Enron database, one for each party. The set of added elements are randomly sampled from the remaining data in the Enron database, and the set of removed elements are randomly selected from the parties datasets (with varying sizes). This setting corresponds to a worst-case scenario for our constructions because both original datasets are equally large.

We measured both the running time and the communication overhead. The results are shown in Table 4. From this table, we clearly observe that our protocols become the fastest when the difference of the size of the update and size of the old set is maximized. This is an expected behavior thanks to the asymmetric nature of the underlying circuit PSI.

## 8 Conclusion

We have proposed new constructions for updatable PSI and updatable cardinality-PSI, improving the current state-of-the-art [4] by 11× to 40× for updatable PSI, and by 14× to 107× for updatable cardinality-PSI in computation time in the LAN setting.

Our work may inspire several avenues of follow-up works. One of these would be to continue the path towards updatable circuit-PSI, even if this would need to come from a different architecture than ours (using four smaller circuit-PSI executions). It could also take shape of proposing schemes for other specific circuits (such as fuzzy PSI, or threshold PSI). Another avenue is to extend the security

model and solution to the malicious case. The main hurdle in that direction consists of how to commit the sets of each individual execution of the intersection without incurring a linear communication and computation cost with respect to the large sets (thus being asymptotically equivalent to the naive solution of re-executing the full PSI). Finally, with respect to the concrete efficiency of our construction, the main bottleneck of our implementation (both in computation time and communication cost) is the execution of OT extension. To improve this concrete efficiency, a better OT extension library could be implemented or silent OT extension [9,10] can be employed to have an even better asymptotic growth. Also, with respect to the implementation, the code can be optimized to reduce the impact of low bandwidth on the protocol. Finally, the code can be further parallelized by using multiple threads per party, instead of only one.

**Acknowledgments.** This work was partially supported by the French National Research Agency (ANR) Project ANR-23-CE39-0009-06 TRUST, the Scientific and Technological Research Council of Türkiye (TÜBİTAK) projects 119E088, 123E462, and the 1515 Frontier Research and Development Laboratories Support Program project 5169902.

## A A Reactive Functionality for Updatable PSI

Given the dynamic nature of updatable PSI, one could alternatively define its ideal functionality as a reactive functionality. That is, a stateful and interactive functionality where the parties can submit new inputs over time and receive updated outputs. We present a simple reactive ideal functionality for updatable PSI in Figure 4. Similarly, one could consider a reactive ideal functionality for the cardinality version of PSI.

We can show that $\mathcal{F}'_{\text{uPSI}}$ and $\mathcal{F}_{\text{uPSI}}$ in section 4 are equivalent in the honest-but-curious setting, namely there is a (trivial) protocol in the $\mathcal{F}_{\text{uPSI}}$-hybrid world that realizes functionality $\mathcal{F}'_{\text{uPSI}}$ and vice versa.

We focus on one direction of the equivalence showing a protocol realizing $\mathcal{F}'_{\text{uPSI}}$. The protocol requires party $P_1$ to store set $X^i$ and intersection $I^i$, and party $P_2$ to store $Y^i$. At each iteration, party $P_1$ (respectively, $P_2$) sets $X^{\text{old}} = X^i$ and sends the tuple $(X^{\text{old}}, X^+, X^-)$ and $I^i$ (respectively, $(Y^{\text{old}}, Y^+, Y^-)$) to the ideal functionality $\mathcal{F}_{\text{uPSI}}$. The party $P_1$ updates the intersection $I^{i+1}$ according to the output of the functionality.

The correctness of the protocol is straightforward to verify. The only point worth clarifying is the leakage received by the adversary: In the ideal functionality $\mathcal{F}_{\text{uPSI}}$, the adversary learns the cardinalities of both the old and the new sets at each update; in contrast, the reactive functionality $\mathcal{F}'_{\text{uPSI}}$ leaks the cardinality of the updated set, only. This difference, however, is not significant. The initial sets $X^0$ and $Y^0$ are trivially of size zero. Furthermore, at each iteration $i$, the adversary already knows the cardinality of the old set from iteration $i-1$. Therefore, the additional leakage in $\mathcal{F}_{\text{uPSI}}$ does not reveal any new information beyond what the adversary already learns during the protocol execution.

**Functionality 6** One-sided Reactive Updatable Private Set Intersection $\mathcal{F}'_{\text{uPSI}}$

At first activation the functionality initializes empty sets $X^0, Y^0 = \emptyset$ associated to $P_1$ and $P_2$ respectively and a counter $i = 0$.

*Inputs.* At each iteration the party $P_1$ sends input $(X^+, X^-)$ with $|X^+|, |X^-| \leq \alpha$ where $X^+, X^-$ are valid updates for $X^i$, and the party $P_2$ inputs $(Y^+, Y^-)$ with $|Y^+|, |Y^-| \leq \alpha$ where $Y^+, Y^-$ are valid updates for $Y^i$.
The ideal functionality sets $X^{i+1} \leftarrow (X^i \setminus X^-) \cup X^+$ and $Y^{i+1} \leftarrow (Y^i \setminus Y^-) \cup Y^+$.

*Outputs.* When both parties have sent their input for the current iteration, the ideal functionality sends to $P_1$ the new intersection $I^{i+1} = X^{i+1} \cap Y^{i+1}$, $P_2$ receives nothing. The ideal functionality increase its counter $i$. The adversary receives $|X^{i+1}|, |Y^{i+1}|$.

Fig. 4: A reactive Ideal functionality for updatable PSI.

Finally, we note that this composition is only possible because we consider the honest-but-curious setting. In the malicious setting, a corrupted party $P_1$ may not maintain a consistent internal state. In particular, it can arbitrarily modify $I^i$ and $X^i$, or it may send invalid updates, because there is no guarantee that $X^- \subseteq X^{\text{old}}$ or that $X^+ \cap X^{\text{old}} = \emptyset$ will hold. Without these conditions, the integrity of the update process cannot be enforced, making the composition insecure in the presence of an actively malicious adversary.

## B Numerical Example of Disjoint Sets

In this section. we give a concrete numerical example of the disjoint sets we compute to facilitate its comprehension. Let $P_1$ own $X^{old} = \{2, 6, 8\}$, and $P_2$ own $Y^{old} = \{2, 5, 6\}$ and as such $I^{old} = \{2, 6\}$. Then let the update be $X^+ = \{1, 5\}$, $Y^+ = \{1, 8\}$, $X^- = \{2, 8\}$ and $Y^- = \{2, 6\}$. Adding it all together, then the partial intersections we compute will be $I_1 = Y^+ \cap ((X^{old} \setminus X^-) \cup X^+) = \{1\}$, $I_2 = X^+ \cap (Y^{old} \setminus Y^-) = \{5\}$, $I_3 = Y^- \cap X^{old} = \{2, 6\}$ and $I_4 = X^- \cap (Y^{old} \setminus Y^-) = \emptyset$. Which in turn give us $I^{new} = (I^{old} \setminus (I_3 \cup I_4)) \cup (I_1 \cup I_2) = \{1, 5\}$, noting that $X^{new} = \{1, 5, 6\}$ and $Y^{new} = \{1, 5, 8\}$.

Now consider a second update, in which $P_1$ adds $X^+ = \{3, 4\}$ and removes $X^- = \{1, 6\}$ while $P_1$ adds $Y^+ = \{2, 4\}$ and removes $Y^- = \{1, 5\}$. In this step, we will then have $X^{old} = \{1, 5, 6\}$, $Y^{old} = \{1, 5, 8\}$ and $I^{old} = \{1, 5\}$ (which are the new sets of the previous step), and the computed sets will be $I_1 = \{4\}$, $I_2 = \emptyset$, $I_3 = \{1, 5\}$ and $I_4 = \emptyset$. This will then give us $I^{new} = (I^{old} \setminus (I_3 \cup I_4)) \cup (I_1 \cup I_2) = \{4\}$, noting that $X^{new} = \{3, 4, 5\}$ and $Y^{new} = \{2, 4, 8\}$.

# C Security Definitions and Proofs

## C.1 Security Definitions

**Definition 2 (Correctness and Secrecy of Secret Sharing).** *Let* $\mathsf{SS} = (\mathsf{SS.SetUp}, \mathsf{SS.Share}, \mathsf{SS.Combine})$ *be a 2-out-of-2 secret sharing scheme.*

- *We say it is* correct *if for any element* $x$ $\Pr[\mathsf{SS.Combine}(s^{(X)}, s^{(Y)}) \neq x] = \mathsf{negl}(\kappa)$, *where* $\mathsf{SS.param} \leftarrow \mathsf{SS.SetUp}(1^\kappa)$ *and* $(s^{(X)}, s^{(Y)}) \leftarrow \mathsf{SS.\ Share}(x)$.
- *We say it is* secret *if for any elements* $x_0, x_1$ *the following distributions are indistinguishable* $s_0^{(X)} \approx s_1^{(X)}$, $s_0^{(Y)} \approx s_1^{(Y)}$ *where* $(s_0^{(X)}, s_0^{(Y)}) \leftarrow \mathsf{SS.Share}(x_0)$ *and* $(s_1^{(X)}, s_1^{(Y)}) \leftarrow \mathsf{SS.Share}(x_1)$.

**Definition 3 (Security of 2PC Protocol).** *Let* $\Pi$ *be a 2PC protocol, let* $View_i^{\Pi}(X;Y)$ *be the view of party* $P_i$ *(including input, random tape and all received messages) of the protocol* $\Pi$ *with inputs* $X$ *by* $P_X$ *and* $Y$ *by* $P_Y$, *and let* $Out_i^{\Pi}(X;Y)$ *be the output of* $P_i$ *in protocol* $\Pi$ *on inputs* $X$ *by* $P_X$ *and* $Y$ *by* $P_Y$. *Then,* $\Pi$ *is said to securely compute an ideal functionality* $\mathcal{F}$ *in the semi-honest model if for any PPT adversary* $\mathcal{A}$ *there exist PPT simulators* $Sim_X$ *and* $Sim_Y$ *such that for all inputs* $x$ *and* $y$

$$View_X^{\Pi}(X;Y), Out_X^{\Pi}(X;Y) \approx_c Sim_X(x, \mathcal{F}_X(X;Y)), \mathcal{F}_Y(X;Y),$$
$$View_Y^{\Pi}(X;Y), Out_Y^{\Pi}(X;Y) \approx_c Sim_Y(x, \mathcal{F}_Y(X;Y)), \mathcal{F}_X(X;Y),$$

*where* $F_i(X;Y)$ *denotes the output of* $P_i$ *for the ideal functionality on inputs* $X, Y$

## C.2 Proof of Lemma 1

*Proof.* First we get that :

$$I^{new} = (X^{new} \cap (Y^{old} \backslash Y^-)) \cup (X^{new} \cap Y^+) \tag{4}$$
$$= ((X^{old} \backslash X^-) \cap (Y^{old} \backslash Y^-)) \cup (X^+ \cap (Y^{old} \backslash Y^-)) \cup (X^{new} \cap Y^+), \tag{5}$$

where at steps 4, 5 we have applied the distributive property between set intersection and set union and the definition of $Y^{new} = (Y^{old} \setminus Y^-) \cup Y^+$, and simarly for $X^{new}$. We note that

$$((X^{old} \backslash X^-) \cap (Y^{old} \backslash Y^-)) = (X^{old} \backslash (X^- \cup Y^-)) \cap (Y^{old} \backslash (X^- \cup Y^-),$$

in fact, any element in $X^{old} \backslash X^-$ but not in $X^{old} \backslash (X^- \cup Y^-)$ is then in $X^{old} \cap Y^-$ (and in particular in $Y^-$) and therefore not in the intersection $(X^{old} \backslash X^-) \cap (Y^{old} \backslash Y^-)$ (and analogously for $Y^{old} \backslash Y^-$). Subbing the equation above into Equation (5) and setting $I_1 := X^{new} \cap I^+$ we have:

$$I^{new} = \big((X^{old} \backslash (X^- \cup Y^-)) \cap (Y^{old} \backslash (X^- \cup Y^-))\big) \cup (X^+ \cap (Y^{old} \backslash Y^-)) \cup I_1$$
$$= (I^{old} \backslash (X^- \cup Y^-)) \cup (X^+ \cap (Y^{old} \backslash Y^-)) \cup I_1 \tag{6}$$
$$= I^{old} \backslash ((X^- \cup Y^-) \cap (X^{old} \cap Y^{old})) \cup (X^+ \cap (Y^{old} \backslash Y^-)) \cup I_1 \tag{7}$$

At step 6 we have applied the right distributive property of set difference over set intersection and $I^{old} = X^{old} \cap Y^{old}$, at step 7 we have used the definition of set difference. For convenience, we set $I_2 := (X^+ \cap (Y^{old} \backslash Y^-))$ and $I^- := ((X^- \cup Y^-) \cap (X^{old} \cap Y^{old}))$ . Thus

$$I^{new} = I^{old} \backslash I^- \cup (I_2 \cup I_1) \tag{8}$$

We focus now on $I^-$:

$$I^- = (X^- \cap (X^{old} \cap Y^{old})) \cup (Y^- \cap (X^{old} \cap Y^{old})) \tag{9}$$

$$= (X^- \cap Y^{old}) \cup (Y^- \cap X^{old}) \tag{10}$$

$$= (X^- \cap (Y^{old} \backslash Y^-)) \cup (Y^- \cap X^{old}) \tag{11}$$

Where at step 9 we have applied the distributive property between set intersection and set union, at step 10 we have used the associative property of set intersection as well as $X^- \subseteq X^{old}$ (and analogously $Y^- \subseteq Y^{old}$), and finally at step 11 we have used that any element in $X^- \cap Y^{old}$ not in $X^- \cap (Y^{old} \backslash X^-)$ must be in $Y^- \cap X^- \subseteq Y^{old} \cap Y^-$ since $Y^- \subseteq Y^{old}$. Setting $I_3 := (X^{old} \cap Y^-)$ and $I_4 := (X^- \cap (Y^{old} \backslash Y^-))$ and plugging the above derivation for $I^-$ in Equation (8), we obtain Equation (2) in the statement of the lemma.

To finish the proof we only need to prove that the sets are disjoint. It is clear that $I_4 = X^- \cap (Y^{old} \backslash Y^-)$ and $I_3 = X^{old} \cap Y^-$ are disjoint because $Y^{old} \backslash Y^-$ and $Y^-$ are disjoint. Then, $I^{old}$ and $I_1 = (X^{new} \cap Y^+)$ are disjoint since $Y^{old} \cap Y^+ = \varnothing$ (analogously for $X^+ \cap (Y \backslash Y^-)$). Finally, $I_1 = (X^{new} \cap Y^+)$ and $I_2 = X^+ \cap (Y^{old} \backslash Y^-)$ are disjoint since $Y^{old} \cap Y^+ = \varnothing$.

### C.3 Proof of Theorem 1

*Proof.* We first show that the protocol is correct. First we notice that the four call to $\mathcal{F}_{\text{cPSI}}$ compute the sets $I_1, \ldots, I_4$ in Lemma 1. Moreover, since $I_1$ and $I_2$ (resp. $I_3$ and $I_4$) are disjoint the output of the second call (resp. first call) to $\mathcal{F}_{\text{CnP}}$ can be indeed parsed as a set. Thus, by Equation (2) of Lemma 1 the set $I^{new}$ computed by $P_1$ is indeed equal to the intersection $X^{new} \cap Y^{new}$.

We now focus on privacy. Consider the following simulator $\mathcal{S}_1$ w.r.t. corrupted $P_1$:

1. Receive in input $X^{old}, X^+, X^-, I^{old}$ and the output $I^{new}$.
2. Sample secret shares of vectors $\mathbf{0}$ of the size $\alpha$, which is the public threshold parameter, let them be $\boldsymbol{s}^{(i)}_{Y^+}, \boldsymbol{s}^{(i)}_{X^+}, \boldsymbol{s}^{(i)}_{Y^-}, \boldsymbol{s}^{(i)}_{X^-}$ with $i \in \{1, 2\}$. The vectors of shares $\boldsymbol{s}^{(1)}_{Y^+}, \boldsymbol{s}^{(1)}_{X^+}, \boldsymbol{s}^{(1)}_{Y^-}, \boldsymbol{s}^{(1)}_{X^-}$ are considered in the simulated view of $P_1$ as result of the interaction with the (simulated) ideal functionality $\mathcal{F}_{cPSI}$.
3. Compute the set $I^+, I^-$ from $I^{old}, I^{new}$ and set a random permutation of the elements in $I^+$ (resp. in $I^-$) as the output of the interaction with the first call (resp. second call) of the ideal functionality $\mathcal{F}_{CnP}$.

The simulator $\mathcal{S}_2$ w.r.t. corrupted $P_2$ is identical but it does not run the step 3.

We can prove the indistinguishability between the simulated and real worlds in two step. First we perform a hybrid argument, using the secrecy property of the secret-sharing scheme. In each hybrid, we replace one secret-sharing instance of the dummy value 0 with one of the real value. We omit the details of the reduction to the secrecy property, since it follows a standard textbook argument. We then arrive at a hybrid in which the first part executes the real protocol running the functionality $\mathcal{F}_{\text{cPSI}}$, while the hybrid still simulates the output of $\mathcal{F}_{\text{CnP}}$ by setting $I^-, I^+$ as in Step 3. This hybrid would diverge from the real world only if the simulated sets $I^+$ or $I^-$ differed from the values $\bar{I}^+$ and $\bar{I}^-$ obtained by reconstructing the shares. However, both $\bar{I}^-$ and $\bar{I}^+$ are valid updates of $I^{old}$ (by Lemma 1), and by definition $I^-$ and $I^+$ are also valid updates of $I^{old}$. Furthermore, by Equation (3) it follows that $\bar{I}^- = I^-$ and $\bar{I}^+ = I^+$.

## D Towards Updatable Circuit-PSI

In this section, we study the challenges of designing a generic updatable circuit PSI and identify some solutions that address these challenges, partially.

### D.1 Challenges

Generic circuit PSI protocols, as mentioned in Section 2, allow computation of arbitrary circuits over the intersection of two private sets. To do so, especially for the case where one wants to evaluate different circuits over one same intersection, such protocols compute secret shares of the intersection, and further execute generic 2PC protocols [6, 20, 54] for the actual circuit to be executed.

In the particular case of updatable circuit PSI, where we propose to execute four intermediate and smaller intersections, deleting elements becomes a point of contention. Indeed, the existing shares of the deleted elements should be deleted/modified. As also noted in [4], this means that the parties will learn which of the existing shares are modified, unless all elements are updated, incurring a cost linear with the size of the large sets (which is undesirable since this would cost as a naive re-execution of circuit PSI).

While in [4], authors argue that this situation results in the leakage of the lifetime of the elements, only, (i.e. when each element being deleted was previously added), we argue that this leakage is much more important. To explain this greater leakage, we need to take a closer look on how circuit PSI applies the generic 2PC protocol to the secret shared intersection: The output of the circuit PSI is a set of secret shares of either some values corresponding to the existence of the elements in the intersection, or zero (for elements not in the intersection). For example, in an unbalanced circuit PSI, each share represents an element of the small set, with the share being one of 0 if the element is not in the intersection and of 1 (or the value of the element itself) if the element appears in the intersection. This means, in particular, that the owner of the small set knows the mapping between its elements and the secret shares. This mapping can be

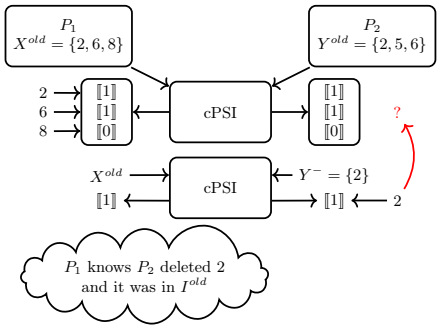



Fig. 5: Inherent leakage in updatable circuit PSI.

necessary for the actual circuit that is to be executed (such as in the case of PSI).

For ease of explanation, in Figure 5, let us assume that both $P_1, P_2$ have shares of the old intersection $X^{old} \cap Y^{old}$ ($\{2, 6\}$ in the example), more concretely, they have $\{s^{(1)}_{old,i}, s^{(2)}_{old,i}\}_{i \in [n_X]}$ which are shares of 0 if the element $x_i$ is not in $X^{old} \cap Y^{old}$ and shares of 1 (or shares of the $x_i$ itself) otherwise. As such, $P_1$ knows the correspondence between the elements of its old dataset and these shares. Now, consider the execution of the intermediate intersection $Y^- \cap X^{old}$ ($\{2\}$ in the example). In this case, the parties obtain $s^{(1)}_{Y^-,i}, s^{(2)}_{Y^-,i}{}_{i \in [\alpha]}$. Moreover, we may assume that $|Y^-| \ll |X^{old}|$. To leverage this unbalance for an efficient protocol, $P_2$ (the owner of $Y^-$) should be given the mapping between the output shares and its set $Y^-$. Then, for every $s^{(1)}_{Y^-,i}, s^{(2)}_{Y^-,i}$ combining to 1, we know that there exist some $j$ such that $s^{(1)}_{old,j}, s^{(2)}_{old,j}$ combine to 1 that needs to be modified (in the example, it would be the first row of the shares of $I^{old}$). However, linking these two shares together leaks too much information, since normally $P_2$ does not know the position of the deleted element in the old intersection and $P_1$ does not normally know the value of the deleted element. Indeed, if such a link is made, $P_2$ will learn when the removed element was added by $P_1$ (if we had several update processes before), but more importantly, $P_1$ recovers the exact value being removed by $P_2$ (through its correspondence).

### D.2 Work Arounds

**Previously Chosen Circuit.** One of the ways to avoid the leakage is something we used throughout this work. Certain particular circuits, if known before-hand, may have their own way of circumventing this issue. For example, the identity circuit (i.e. standard PSI) avoids this since the leakage is already covered by the ideal functionality. Similarly, the cardinality circuit avoids matching shares by aggregating everything with the addition. However, having a fixed circuit to compute defeats the purpose of constructing *generic* updatable circuit PSI.

**Only Addition.** Another way, which was already explored in [4], would be to allow only addition of elements in the updates. Since the leakage comes from identifying the shares of the deleted elements in the old intersection, by forbidding deletions, the leakage is circumvented. Note that constructing an updatable circuit PSI with only addition is trivial from our construction of updatable PSI. By executing the steps of Protocol 1 without deletions, and only using $\mathcal{F}_{\text{cPSI}}$ over $X^+ \cap Y^{old}$ and $Y^+ \cap X^{new}$, we obtain a protocol that correctly and securely executes updatable circuit PSI only with addition (correctness comes directly from the underlying cPSI protocol and security is trivial from Theorem 1). For example, in the case of spam filters, the blacklist is only increased by adding new spam mails and the mails being received in the inbox can be considered to only increase (since deletions are not checked against the spam filter).

**One Party Updates.** A final alternative to sidestep the leakage is by only allowing one of the parties to update their dataset. Note that the leakage only appears when two different parties have access to (different) mappings; so if only one party has access to the mappings (i.e. only one party can make updates), the leakage is mostly avoided. The construction would work at a high level as follows. Assuming $P_1$ is the party allowed to update its dataset, the protocol would first run the intersections $X^+ \cap Y^{old}$ and $X^- \cap Y^{old}$, and then $P_1$ would tell $P_2$ which shares should be updated. Since $P_2$ never has access to any mapping, the only leakage is for $P_2$ knowing the lifetime of elements without knowing what exact values have this lifetime, which we argue is a reasonable leakage to allow. Such a situation can appear, for example, in the case of secure password breach alerts, where the credentials do not change, but the data found in the deep web is constantly updated.

## E Implementation Optimization

In this section, we discuss the optimizations made our protocol $\Pi_{\text{uPSI-card}}$ on our implementation due to the fact of using the specific unbalanced circuit PSI protocol from [31] as our building block $\Pi_{\text{cPSI}}$.

Let us first give a high level overview of the construction in [31]. For ease of notation, let us assume that the party $P_1$ has the big dataset and $P_2$ has the small dataset. First of all, $P_2$ encodes its dataset into a cuckoo table and $P_1$ encodes its dataset into a hash table, both with the same hash functions. Note that every row of the cuckoo table has at most one element while each row of the hash table will have potentially many elements. Then, by using a batched Oblivious Programmable Pseudo-Random Function (OPPRF) based on garbled bloom filters and oblivious transfer proposed in [31] as well as the secure equality test proposed in [17], they output to $P_2$ for every row $i$ of the cuckoo table $Enc_X(b)$, a (homomorphic) ciphertext encrypted by $P_1$ where $b = 0$ if the element in row $i$ of the cuckoo table is not in row $i$ of the hash table and $b = 1$ otherwise. Next, through homomorphic operations, $P_2$ computes $Enc_X(d_b)$ for some predefined $d_0, d_1$. Finally, all the ciphertexts are sent back to $P_1$, who decrypts them to obtain $d_b$ for every row in the cuckoo table.

In the application to our updatable private set intersection protocols, we will have $d_0, d_1$ be secret shares, in the case of cardinality shares of either 0 or 1 and in the standard case shares of either 0 or the element in the intersection. The optimization comes from the realization that after the OPPRF and secure equality test, the amount of ciphertexts of 1 already reflects the cardinal. As such in our implementation first we add all the ciphertexts corresponding to the rows of the cuckoo table and then homomorphically add a random value to convert it to a share of the cardinal of the intersection. The improvement is then both in computation time, given that we perform less homomorphic operations as well as less decryptions, and in communication cost, since we send only one ciphertext instead of one per row of the cuckoo table.

## References


1. Abadi, A., Dong, C., Murdoch, S.J., Terzis, S.: Multi-party updatable delegated private set intersection. In: Financial Cryptography and Data Security. pp. 100–119 (2022). https://doi.org/10.1007/978-3-031-18283-9_6
2. Agarwal, A., Cash, D., George, M., Kamara, S., Moataz, T., Singh, J.: Updatable private set intersection from structured encryption. IACR Communications in Cryptology **2**(4) (2026). https://doi.org/10.62056/av4fsgbmo
3. van Baarsen, A., Stevens, M.: Amortizing circuit-PSI in the multiple sender/receiver setting. IACR Communications in Cryptology **1**(3) (2024). https://doi.org/10.62056/a0fhsgvtw
4. Badrinarayanan, S., Miao, P., Shi, X., Tromanhauser, M., Zeng, R.: Updatable private set intersection revisited: Extended functionalities, deletion, and worst-case complexity. In: Advances in Cryptology – ASIACRYPT 2024. pp. 200–233 (2025). https://doi.org/10.1007/978-981-96-0938-3_7
5. Badrinarayanan, S., Miao, P., Xie, T.: Updatable private set intersection. Proceedings on Privacy Enhancing Technologies **2022**(2), 378–406 (2022). https://doi.org/10.2478/popets-2022-0051
6. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. p. 1–10. STOC '88 (1988). https://doi.org/10.1145/62212.62213
7. Benattar, G., Cornejo, M., Leontiadis, I., Poumeyrol, M., Shlomovits, O., Varlakov, D.: Paillier, 0.4.3 edn. (2022), https://crates.io/crates/kzen-paillier
8. Berke, A., Bakker, M., Vepakomma, P., Larson, K., Pentland, A.S.: Assessing disease exposure risk with location data: A proposal for cryptographic preservation of privacy (2020), https://arxiv.org/abs/2003.14412
9. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round ot extension and silent non-interactive secure computation. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. p. 291–308. CCS '19 (2019). https://doi.org/10.1145/3319535.3354255
10. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent ot extension and more. In: Advances in Cryptology – CRYPTO 2019. pp. 489–518 (2019). https://doi.org/10.1007/978-3-030-26954-8_16

11. Chandran, G.R., Schneider, T., Stillger, M., Weinert, C.: Concretely efficient private set union via circuit-based psi. In: Proceedings of the 20th ACM Asia Conference on Computer and Communications Security. p. 149–162. ASIA CCS '25 (2025). https://doi.org/10.1145/3708821.3710839
12. Chandran, N., Gupta, D., Shah, A.: Circuit-psi with linear complexity via relaxed batch opprf. Proceedings on Privacy Enhancing Technologies **2022**(1), 353–372 (2022). https://doi.org/10.2478/popets-2022-0018
13. Chase, M., Ghosh, E., Poburinnaya, O.: Secret-shared shuffle. In: Advances in Cryptology – ASIACRYPT 2020. pp. 342–372 (2020). https://doi.org/10.1007/978-3-030-64840-4_12
14. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious prf. In: Advances in Cryptology – CRYPTO 2020. pp. 34–63 (2020). https://doi.org/10.1007/978-3-030-56877-1_2
15. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled psi from fully homomorphic encryption with malicious security. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. p. 1223–1237. CCS '18 (2018). https://doi.org/10.1145/3243734.3243836
16. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 1243–1255. CCS '17 (2017). https://doi.org/10.1145/3133956.3134061
17. Ciampi, M., Orlandi, C.: Combining private set-intersection with secure two-party computation. In: Security and Cryptography for Networks. pp. 464–482 (2018). https://doi.org/10.1007/978-3-319-98113-0_25
18. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Advances in Cryptology - EUROCRYPT 2004. pp. 1–19 (2004). https://doi.org/10.1007/978-3-540-24676-3_1
19. Ghosh, S., Nilges, T.: An algebraic approach to maliciously secure private set intersection. In: Advances in Cryptology – EUROCRYPT 2019. pp. 154–185 (2019). https://doi.org/10.1007/978-3-030-17659-4_6
20. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. p. 218–229. STOC '87 (1987). https://doi.org/10.1145/28395.28420
21. Goldreich, O.: Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press, USA, 1st edn. (2009)
22. Google: Private intersection-sum protocols with applications to attributing aggregate ad conversions, https://research.google/pubs/pub51026/
23. Hao, M., Liu, W., Peng, L., Li, H., Zhang, C., Chen, H., Zhang, T.: Unbalanced Circuit-PSI from oblivious Key-Value retrieval. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 6435–6451 (Aug 2024), https://www.usenix.org/conference/usenixsecurity24/presentation/hao-meng-unbalanced
24. Harchandani, Lovesh: Oblivious Transfer (OT), Oblivious Transfer Extensions (OTE) and multi-party protocols based on that, 0.11.0 edn. (2025), https://crates.io/crates/oblivious_transfer_protocols
25. Hazay, C.: Oblivious polynomial evaluation and secure set-intersection from algebraic prfs. In: Theory of Cryptography. pp. 90–120 (2015). https://doi.org/10.1007/978-3-662-46497-7_4
26. Hetz, L., Schneider, T., Weinert, C.: Scaling mobile private contact discovery to billions of users. In: Computer Security – ESORICS 2023: 28th European Symposium on Research in Computer Security, 2023, Proceedings, Part I. p. 455–476 (2023). https://doi.org/10.1007/978-3-031-50594-2_23

27. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: 19th Annual Network and Distributed System Security Symposium, NDSS 2012. The Internet Society (2012), https://www.ndss-symposium.org/ndss2012/private-set-intersection-are-garbled-circuits-better-custom-protocols
28. Kales, D., Rechberger, C., Schneider, T., Senker, M., Weinert, C.: Mobile private contact discovery at scale. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 1447–1464 (Aug 2019), https://www.usenix.org/conference/usenixsecurity19/presentation/kales
29. Kamara, S., Mohassel, P., Raykova, M., Sadeghian, S.S.: Scaling private set intersection to billion-element sets. In: Financial Cryptography and Data Security. pp. 195–215 (2014). https://doi.org/10.1007/978-3-662-45472-5_13
30. Karakoç, F., Küpçü, A.: Linear complexity private set intersection for secure two-party protocols. In: Cryptology and Network Security. pp. 409–429 (2020). https://doi.org/10.1007/978-3-030-65411-5_20
31. Karakoç, F., Küpçü, A.: Enabling two-party secure computation on set intersection. IEEE Transactions on Dependable and Secure Computing pp. 1–12 (2025). https://doi.org/10.1109/TDSC.2025.3561472
32. Kiss, Á., Liu, J., Schneider, T., Asokan, N., Pinkas, B.: Private set intersection for unequal set sizes with mobile applications. Proceedings on Privacy Enhancing Technologies **2017**(4), 177–197 (2017). https://doi.org/10.1515/popets-2017-0044
33. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious prf with applications to private set intersection. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. p. 818–829. CCS '16 (2016). https://doi.org/10.1145/2976749.2978381
34. Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: Practical multi-party private set intersection from symmetric-key techniques. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 1257–1272. CCS '17 (2017). https://doi.org/10.1145/3133956.3134065
35. Ling, G., Tang, P., Qiu, W.: Efficient updatable PSI from asymmetric PSI and PSU. Cryptology ePrint Archive, Paper 2024/1712 (2024), https://eprint.iacr.org/2024/1712
36. Mahdavi, R.A., Lukas, N., Ebrahimianghazani, F., Humphries, T., Kacsmar, B., Premkumar, J., Li, X., Oya, S., Amjadian, E., Kerschbaum, F.: Pepsi: practically efficient private set intersection in the unbalanced setting. In: Proceedings of the 33rd USENIX Conference on Security Symposium. SEC '24 (2024), https://www.usenix.org/conference/usenixsecurity24/presentation/mahdavi
37. Marlinspike, M.: Technology preview: Private contact discovery for signal, https://signal.org/blog/private-contact-discovery/
38. Meadows, C.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: 1986 IEEE Symposium on Security and Privacy. pp. 134–134 (1986). https://doi.org/10.1109/SP.1986.10022
39. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology – EUROCRYPT 1999. p. 223–238 (1999). https://doi.org/10.1007/3-540-48910-X_16
40. Patrick Nepper, Kiran C. Nair, V.S., Khaneja, V.: Better password protections in chrome, https://security.googleblog.com/2019/12/better-password-protections-in-chrome.html

41. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Spot-light: Lightweight private set intersection from sparse ot extension. In: Advances in Cryptology – CRYPTO 2019. pp. 401–431 (2019). https://doi.org/10.1007/978-3-030-26954-8_13
42. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Psi from paxos: Fast, malicious private set intersection. In: Advances in Cryptology – EUROCRYPT 2020. pp. 739–767 (2020). https://doi.org/10.1007/978-3-030-45724-2_25
43. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on ot extension. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 797–812 (Aug 2014), https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-pinkas.pdf
44. Raghuraman, S., Rindal, P.: Blazing fast psi from improved okvs and subfield vole. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 2505–2517. CCS '22 (2022). https://doi.org/10.1145/3548606.3560658
45. Rindal, P., Schoppmann, P.: Vole-psi: Fast oprf and circuit-psi from vector-ole. In: Advances in Cryptology – EUROCRYPT 2021. pp. 901–930 (2021). https://doi.org/10.1007/978-3-030-77886-6_31
46. RustCrypto development team: RustCrypto: Advanced Encryption Standard (AES), 0.8.4 edn. (2024), https://crates.io/crates/aes
47. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (Nov 1979). https://doi.org/10.1145/359168.359176
48. Son, Y., Jeong, J.: Psi with computation or circuit-psi for unbalanced sets from homomorphic encryption. In: Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security. p. 342–356. ASIA CCS '23 (2023). https://doi.org/10.1145/3579856.3582817
49. Song, X., Yin, D., Bai, J., Dong, C., Chang, E.C.: Secret-shared shuffle with malicious security. In: Network and Distributed System Security Symposium 2024 (NDSS 2024) (01 2024). https://doi.org/10.14722/ndss.2024.24021
50. Trieu, N., Shehata, K., Saxena, P., Shokri, R., Song, D.: Epione: Lightweight contact tracing with strong privacy (2020), https://arxiv.org/abs/2004.13293
51. Wang, R., Zhou, J., Cao, Z., Dong, X., Raymond Choo, K.K.: Updatable private set intersection with forward privacy. IEEE Transactions on Information Forensics and Security **19**, 8573–8586 (2024). https://doi.org/10.1109/TIFS.2024.3461475
52. Wikipedia: Exposure notification, https://en.wikipedia.org/wiki/Exposure_Notification
53. Yang, Y., Liang, X., Song, X., Dong, Y., Huang, L., Ren, H., Dong, C., Zhou, J.: Maliciously secure circuit private set intersection via spdz-compatible oblivious PRF. Proceedings on Privacy Enhancing Technologies **2025**(2), 680–696 (2025). https://doi.org/10.56553/popets-2025-0082
54. Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986). pp. 162–167 (1986). https://doi.org/10.1109/SFCS.1986.25