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

Ferran Alborch
EURECOM
Sophia Antipolis, France
ferran.alborch@eurecom.fr

Tom Chauvier
EURECOM
Sophia Antipolis, France
tom.chauvier@eurecom.fr

Antonio Faonio
EURECOM
Sophia Antipolis, France
antonio.faonio@eurecom.fr

Alexandre Fontaine
EURECOM
Sophia Antipolis, France
alexandre.fontaine@eurecom.fr

Ferhat Karakoç
Ericsson Research
İstanbul, Türkiye
ferhat.karakoc@ericsson.com

Alptekin Küpçü
Koç University
İstanbul, Türkiye
akupcu@ku.edu.tr

Camille Malek
EURECOM
Sophia Antipolis, France
camille.malek@tutamail.com

Melek Önen
EURECOM
Sophia Antipolis, France
melek.onen@eurecom.fr

Abstract*—Private Set Intersection (PSI) has been widely studied, deployed, and demonstrated through various real-life use cases such as mobile private contact discovery, privacy-preserving contact tracing, etc. Nevertheless, the majority of existing solutions typically assume that the underlying datasets are static and require a fresh execution of PSI at each time the datasets are updated over time. In this work, similar to a recent solution by Badrinaryanan et. al' (ASIACRYPT 2024), 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 two privacy leakages and further propose to mitigate them thanks to the use of circuit PSIs, which are variants of PSI protocols that instead of outputting the resulting intersection, output the secret shares of the intersection and nothing more, combined with secure shuffling when needed. We construct a generic framework for PSI over updated sets which can use any circuit-PSI. Additionally, we show that this framework can easily be extended to a protocol that outputs the cardinality of the intersection instead of the intersection, itself. Finally, we provide an in-depth discussion on the feasibility of circuit PSI over updated sets, with the main challenges to overcome and solutions for some particular cases. Our solutions are implemented in Rust and their performance is compared with the state of the art, achieving an improvement of $11\times$ to $40\times$ in updatable PSI and $14\times$ to $107\times$ in updatable cardinality PSI in computation time. The proposed framework is also demonstrated through a real-life use case, namely, a spam detection filter.

***Index Terms*—private set intersection, updated or dynamic datasets, extended functionalities**

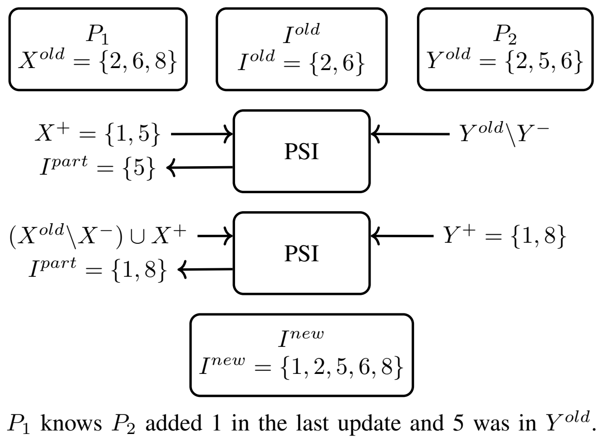
1. Introduction

Private Set Intersection (PSI) enables two mutually distrusting parties, each of them holding some private datasets, to compare these data sets and disclose only their common records, and nothing more. This functionality excels in various use cases, such as mobile private contact discovery[1] [1], [2], privacy-preserving contact tracing[2] [3], [4], online advertising[3] [5], or privacy breach detection[4]. Researchers have investigated various challenges such as scalability [6], balanced and unbalanced settings with respect to the size of the datasets or the resources of the actual parties [7], the extension to the multi-party setting [8], 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) [9].

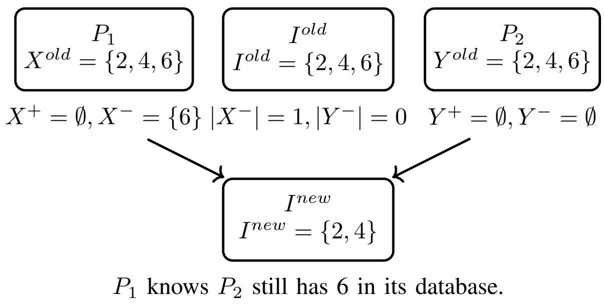
One aspect that has not yet been investigated deeply is the case where datasets are not static, i.e. datasets are modified over time. Indeed, real datasets are frequently updated as new data is generated or collected and some data might have become useless or have expired. In this context, the computed intersections or other further computations over these intersections can become obsolete rapidly. While there exist some solutions that tackle this problem that aim at taking advantage of the existing output and therefore not re-executing the PSI or circuit-PSI protocol from scratch [10]–[14], we observe that these still suffer from privacy leakages (as seen in Figure 1 and explained in Section 2).

Contributions. In this paper we identify and study these leakages, which arise when one considers both additions and deletions in the datasets, and further propose new constructions for PSI and cardinality of PSI over updated sets in the honest-but-curious model. Similar to [11], these 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. Compared to the related work, our solutions support both addition and deletion of elements and features

1. https://signal.org/blog/private-contact-discovery/
2. https://openmined.org/blog/private-set-intersection/ and https://covid19.apple.com/contacttracing
3. https://research.google/pubs/pub51026/
4. https://security.googleblog.com/2019/12/better-password-protections-in-chrome.html



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



P_1
$X^{old} = \{2,4,6\}$
I^{old}
$I^{old} = \{2,4,6\}$
P_2
$Y^{old} = \{2,4,6\}$
$X^+ = \emptyset, X^- = \{6\}$ $|X^-| = 1, |Y^-| = 0$ $Y^+ = \emptyset, Y^- = \emptyset$
I^{new}
$I^{new} = \{2,4\}$
P_1 knows P_2 still has 6 in its database.

(b) Leakage concerning the size of the update.

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

better performance. Notably, we obtain an improvement of 11× to 40× in updatable PSI and 14× to 107× in updatable cardinality PSI in computation time. Another important advantage of our solution is that it supports the client-server setting where only one client receives the ultimate output. We also provide an in-depth discussion of the feasibility of general updatable circuit PSI, with the main challenges to overcome and solutions for some particular cases.

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 apart from this intersection. For some applications, however, even leaking the intersection itself might be too much. For example, if the use case requires computing only the size of the intersection (e.g., when computing counting queries over shared data), outputting the elements of the intersection could be considered as a leakage. A particular kind of PSI is then used, called **Circuit-PSI**, where the parties obtain secret shares of the intersection. Then these shares can be used within a 2PC protocol to obtain the intended function output over the intersection. **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 majority of existing updatable PSI or updatable cardinality PSI solutions split the sets into significantly smaller ones and repetitively execute some operations over these smaller sets (such as the set of added or removed elements on the one hand side and the new set on the other side).

In the same line, we investigate the execution of 4 PSIs among four disjoint sets that were also proposed in a previous work by Badriyanan et al. [11]. These disjoint sets are explained in depth in Section 5.2. This distribution prevents the double counting of elements and therefore ensures correctness of PSI and its extended functionalities.

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

Indeed, consider the example in Figure 1a illustrating an updatable PSI scenario. In this figure, we denote the old sets with X^{old}, Y^{old}, the elements added with X^+, Y^+, and the elements removed from the respective sets with X^-, Y^-. Thus we have $X^{new} = (X^{old} \setminus X^-) \cup X^+$ and similarly $Y^{new} = (Y^{old} \setminus Y^-) \cup Y^+$. 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.

To illustrate such a leakage with a real-life use case, we consider the use of privacy-preserving contact tracing during the COVID-19 pandemic. Such applications compute the PSI of the locations of pairwise users they were in contact with [3], [4]. 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 four intermediate operations executed on smaller sets need to be protected. The natural protocol for this is circuit-PSI by the inherent protection given by the (secret shared) output.

Nevertheless, we identify that this new proposal still suffers from two additional categories of leakages.

- **Leakage of the size of the input sets to circuit-PSIs:** While the output of the circuit-PSI becomes oblivious, an adversary can still exploit the information about the inputs to these protocols. Consider the example in Figure 1b, in a PSI scenario. 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 the update sizes. 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.
- **Leakage of reconstructed outputs from circuit-PSIs:** To compute the new intersection, one of the parties will reconstruct using the output of the circuit PSIs. As previously mentioned (refer to the example in 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. As such, the party reconstructing intermediate outputs should not be able to discover which element is the output of which circuit-PSI (among the four ones). 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 the leakage of the size of the input sets, we propose to pad the sets for the four circuit-PSIs and obtain a fixed size of α elements for each of them. On the other hand, to prevent a party from discovering the specific outputs of a particular circuit-PSI, we propose to use an oblivious permutation before the reconstruction of the actual elements.

3. Related Work

Private Set Intersection. The first PSI construction was proposed by Meadows in [15], 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 [16]–[18], polynomials over finite fields [19]–[21], homomorphic encryption [22], [23], or oblivious key-value stores and vector oblivious linear evaluation [24]–[26]. 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 [7], [27], [28] 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 [9], 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 [7], [25], [29]–[32] to be further used in any upcoming circuit, with linear computation and communication. The most generalized version encompassing the rest is that of [33], 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. [34] 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. [35] 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. [10]. 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 [11], they give one-sided solutions for arbitrary deletions as well as tackling some extended functionalities like cardinality or payload sum 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

TABLE 1: Comparison of updatable PSI schemes with extended functionalities, where α denotes the size of the updates and n the size of the sets.

	Functionality	Updates	Communication Complexity	Output
[10]	PSI	Addition	$O(\alpha \cdot \log n)$	One-sided
[11]			$O(\alpha \cdot \log n)$	One-sided
[11]		Addition & Deletion	$O(\alpha \cdot \log^2 n)$	One-sided
[14]			$O(\alpha \cdot \log n)^{\dagger}$	Two-sided
[12]			$O(\alpha \cdot \log n^2)$	Two-sided
[13]			$O(\alpha \cdot \log n + \sqrt{n})$	Two-sided
Section 5			$O(\alpha \cdot \log n + \alpha \cdot \log \alpha)$	One-sided
[11]	Cardinality PSI	Addition	$O(\alpha \cdot \log n)$	One-sided
[11]		Addition & Deletion	$O(\alpha \cdot \log^2 n)$	One-sided
Section 6			$O(\alpha \cdot \log n)$	One-sided

†In [14] 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 the added elements is α rather than $\log \alpha$.

necessity. It also does not solve the leakage of the size of the updates.

Wang et al. [14] propose a construction based on sparse PIR, but leaks both the intermediate intersections and the size of the updates. Agarwal et al. [12] construct an updatable PSI scheme based on a novel dynamic structured encryption scheme, but leak the size of the update. Ling et al. [13] give a generic construction based on unbalanced PSI (where one party has a small set) and private set union, but they leak the size of the updates. All these constructions only consider two-sided outputs.

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

4.1. Notation and Basic Definitions

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}. Given the sets X^{old}, X^+, X^-, we uniquely identify $X^{new} = (X^{old} \setminus X^-) \cup X^+$, 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 κ and λ the computational and statistical security parameters, respectively. We identify the ideal functionality for a task F with $\mathcal{F}_{\mathrm{F}}$, and with Π_{F} we denote the corresponding protocol realizing such a 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 the 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 Π_{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 $\mathcal{F}_{\mathrm{G}}$.

In section A, we recall the composition theorem [36, Theorem 7.3.3], which states that if a protocol Π_{F} securely realizes $\mathcal{F}_{\mathrm{F}}$ in the $\mathcal{F}_{\mathrm{G}}$-hybrid model and a protocol Π_{G} securely realizes $\mathcal{F}_{\mathrm{G}}$, then we can construct a new composed protocol Π'_{F} that executes Π_{F}, and whenever Π_{F} invokes the trusted functionality $\mathcal{F}_{\mathrm{G}}$, the call is replaced by an execution of Π_{G}. The composed protocol Π'_{F} securely realizes $\mathcal{F}_{\mathrm{F}}$.

4.2. Secret Sharing

A *t-out-of-n* secret sharing scheme [37], 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, which are defined 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.*

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

Secret sharing schemes have two main properties: *correctness* and *secrecy*, which are formally defined in Appendix A. Basically, correctness means when sharing and combining are done honestly, the correct secret would be recovered. Secrecy means that when less than t 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 a 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 the 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 the intersection $I = X \cap Y$, P_2 receives nothing. The adversary receives $|X|, |Y|$.

are combined, they do not reveal the secret s. In this work we will focus on arithmetic secret shares.

4.3. Combine and Permute

The Combine and Permute (CnP) [38], [39] algorithm is a cryptographic primitive intended to include the privacy enhancement of shuffling into secret sharing schemes, so as to be more easily used as building blocks for more complicated protocols. The main idea is that when shares are being reconstructed, there is a link between the share and the reconstructed value, which, in larger protocols with shares being obtained from different subroutines, may leak unwanted information. Then, 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 [40].

We have P_1 (who will receive the combined shares) who inputs a set of m secret shares $\{s_i^{(1)}\}_{i\in[m]}$ and P_2 who 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 the permutation π, $\pi(\{\mathsf{SS.Combine}(s_i^{(1)}, s_i^{(2)})\}_{i\in[m]})$ to P_1. The ideal functionality is given in Functionality 1.

4.4. Circuit PSI (cPSI)

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 then 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 elements, while others have secret shares of 1 with one of the parties having a correspondence to their dataset.

More in detail, P_1 inputs a set X of size n_X and P_2 inputs a 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.

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 the 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 $|X^+|, |X^-| \le \alpha$ 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 $|Y^+|, |Y^-| \le \alpha$ 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 $|X^{old}|, |Y^{old}|$.

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^-, and 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.

5.2. Construction of Disjoint Sets

Let P_1 be a party owning the sets X^{old}, X^+, X^- and P_2 be an entity owning the sets Y^{old}, Y^+, Y^- such that X^+, X^- (resp. Y^+, Y^-) are valid updates for X^{old} (resp. Y^{old}). In general, we assume $|X^+|, |X^-| \ll |X^{old}|$ (and analogously for Y). The end goal is to compute the 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 $I^{old} = X^{old} \cap Y^{old}$, and to do so by only computing a constant number of partial intersections without leaking their results, where at least one of the sets being involved is small.

The main candidates for small sets in these update intersections are the four update sets $X^{+}, X^{-}, Y^{+}, Y^{-}$. Note that at least four intermediate intersections will be needed, since each party has access to only two of these sets (their added and deleted elements), and they are disjoint ($X^{+} \cap X^{-} = \emptyset$ and $Y^{+} \cap Y^{-} = \emptyset$), which means that the only non-empty set constructible from them is their union, that will not work to compute the updated intersection since added and deleted elements will have a different impact on the updated intersection.

The first intuition is to use symmetric intermediate intersections for both parties: the added elements by each party $X^{+} \cap Y^{new}$ and $Y^{+} \cap X^{new}$ as well as the deleted elements by each party $X^{-} \cap Y^{old}$ and $Y^{-} \cap X^{old}$, and then compute

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

However, note that there will be elements appearing in more than one of these partial intersections. For example $X^{+} \cap Y^{+}$ appears in both $X^{+} \cap Y^{new}$ and $Y^{+} \cap X^{new}$. This raises some issues. Firstly, if the union of the partial intersection is not done obliviously, this setup clearly leaks the elements in $X^{+} \cap Y^{+}$ and $X^{-} \cap Y^{-}$, which is an unauthorized leakage (for a specific real world example of this type of leakage we refer to Section 2). Secondly, when computing other functionalities related to PSI, such as computing the cardinality of the intersection, the repeated elements will be counted twice, harming the correctness of the scheme. Therefore, our goal is to obtain I^{+} (resp. I^{-}) as *disjoint union* of two intersections.

To achieve these four disjoint intersections, we look at the elements in $X^{+} \cap Y^{+}$ and $X^{-} \cap Y^{-}$ and ensure that they only are counted in one of the intersections. That means that since $X^{+} \cap Y^{+}$ is already in $X^{+} \cap Y^{new}$, we need to remove them from the other intersection giving us $Y^{+} \cap X^{old} \backslash X^{-}$, and similarly since $X^{-} \cap Y^{-}$ is already in $Y^{-} \cap X^{old}$, we need to remove them from the other intersection giving us $X^{-} \cap Y^{old} \backslash Y^{-}$. Putting it all together we get

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

where $\uplus$ denotes the disjoint union of sets, namely, $A \uplus B = A \cup B$ if $A \cap B = \emptyset$, otherwise it defines their coproduct. For a visual representation of this construction we refer to Figure 2. 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.

Formally, we get the following lemma.

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

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

Then

$$X^{new} \cap Y^{new} = (I^{old} \setminus (I_3 \cup I_4)) \cup (I_1 \cup I_2) \qquad (1)$$

$$I_1 \cap I_2 = I_3 \cap I_4 = I^{old} \cap I_1 = I^{old} \cap I_2 = \emptyset \qquad (2)$$

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

We refer to Appendix A 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

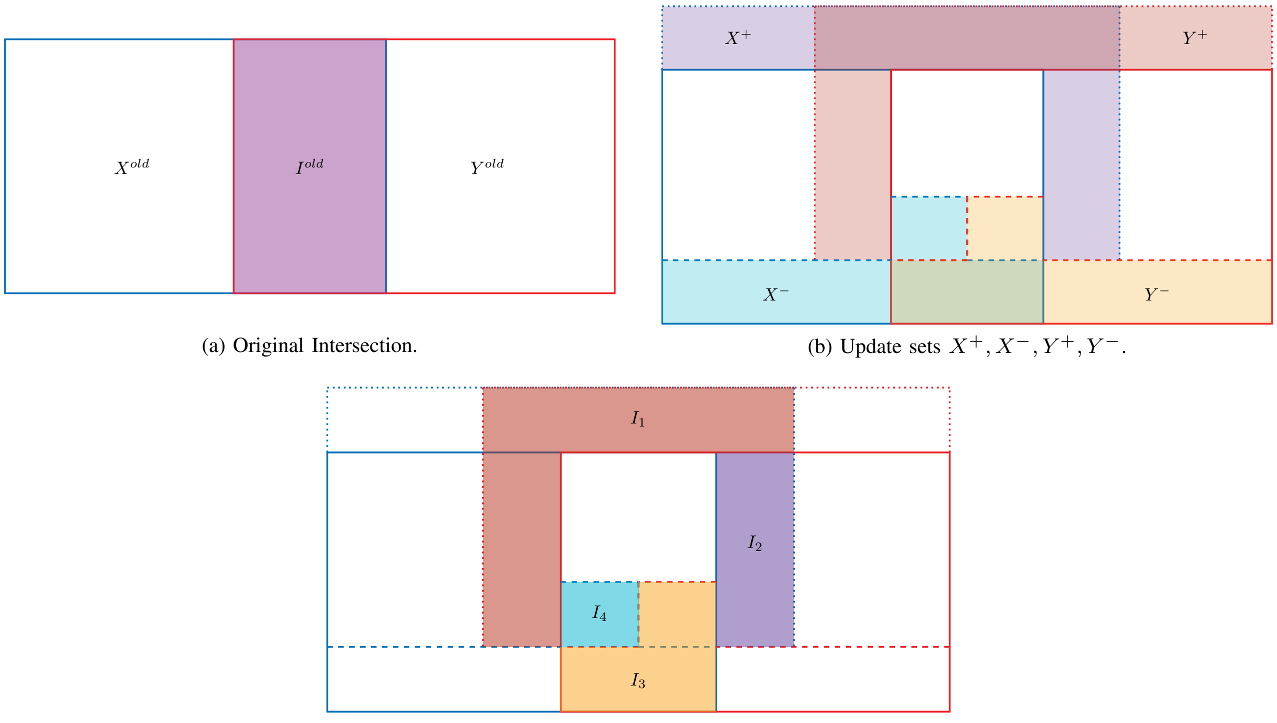
To compute the updated intersection while improving the performance with respect to the naive PSI solution, our protocol calls a protocol securely realizing the ideal functionality $\mathcal{F}_{cPSI}$ with the PSI mode four disjoint times with the sets $X^{+}, X^{-}, Y^{+}, Y^{-}$. Since the size of the updates are much smaller than the sizes of X and Y, performing four calls to an unbalanced PSI is faster and less expensive 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 α. Thus, P_1 ... Similarly P_2 ... We also discuss how to pick α below.

Then, 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. For a concrete explanation of the constructions on these sets we refer to Appendix 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 Figure 1.

On the selection of α. For practical scenarios, we note that there is a tradeoff in the choice of the parameter α. On one hand, if α is too small, the parties may be forced to run PSI unnecessarily (for instance, when the application layer of the protocol does not require knowledge of the updated cardinality I^{new}), thereby risking the exposure of data before it is actually needed. On the other hand, since



(a) Original Intersection. (b) Update sets X^+, X^-, Y^+, Y^-. (c) Disjoint intersections we compute.

Figure 2: Representation of the sets involved in updatable private set intersection. As a concrete numerical example, let Figure 2a represent $X^{old} = \{2, 6, 8\}$, $Y^{old} = \{2, 5, 6\}$ and as such $I^{old} = \{2, 6\}$. Then let the update sets in Figure 2b represent $X^+ = \{1, 5\}$, $Y^+ = \{1, 8\}$, $X^- = \{2, 8\}$ and $Y^- = \{2, 6\}$. Adding it all together, then the sets we compute represented in Figure 2c will be $I_1 = Y^+ \cap ((X^{old} \backslash X^-) \cup X^+) = \{1\}$, $I_2 = X^+ \cap (Y^{old} \backslash Y^-) = \{5\}$, $I_3 = Y^- \cap X^{old} = \{2, 6\}$ and $I_4 = X^- \cap (Y^{old} \backslash Y^-) = \emptyset$. Which in turn give us $I^{new} = (I^{old} \setminus (I_3 \cup I_4)) \cup (I_1 \cup I_2) = \{1, 5\}$.

the protocol's complexity grows proportionally with α, selecting a value that is too large is likewise undesirable.

One possible solution that allows keeping α small without exposing data before it is needed is to maintain the data in secret-shared form until reconstruction is required, while in the background continuing to run the protocol Π_{uPSI} in Protocol 1 with a relatively small α.

In particular, we can partition the update sets into eras. Let $X^+_{(i)}, X^-_{(i)}$ (resp. $Y^+_{(i)}, Y^-_{(i)}$) denote the valid updates for X^{old} (resp. Y^{old}) during era i (for example, the last 10 minutes of updates), where $i \in [t]$. Define $X^- = \bigcup_i X^-_{(i)}$ and $X^+ = \bigcup_i X^+_{(i)}$ (and analogously for Y^- and Y^+) as the sets considered when the new intersection is required (for example, every hour, so $t = 6$). We can then modify the protocol to execute steps (1) and (2) repeatedly for t eras, and then execute step (3) (or alternatively, run step (3) only upon request.). At that point, I^- is computed using the CnP sub-protocol and the shares

$$(\boldsymbol{s}^{(j)}_{Y^-_{(1)}} \| \boldsymbol{s}^{(j)}_{X^-_{(1)}} \| \dots \| \boldsymbol{s}^{(j)}_{Y^-_{(t)}} \| \boldsymbol{s}^{(j)}_{X^-_{(t)}}), \quad j \in \{1, 2\}$$

and I^+ are computed analogously.

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 it is not possible to have that string in items of the set domain.

5.4. Complexity Analysis.

For ease of notation, let us assume $n = |X| = |Y|$. The complexity of the our updatable PSI protocol depends on the complexity of the underlying building blocks, namely the protocols Π_{cPSI} and Π_{CnP} instantiating their respective ideal functionalities. Indeed, the communication cost is computed as 4 times the communication cost of Π_{cPSI} in step 2, and 2 times the cost of Π_{CnP} in step 3. Considering an asymmetric circuit PSI like [7] with a communication complexity of $O(\alpha \cdot \log(N/\alpha))$ and a CnP like [41] 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 Π_{cPSI} in step 1 followed by the 2 parallel executions of Π_{CnP} and finished (for P_1) by the reconstruction of I^{new}. Using, for example, the cPSI from [27] leveraging the linear complexity in the large set, we could get an asymptotic computation complexity of $O(n + \alpha \cdot \log \alpha)$. Note that a large part of this is for offline computation.

Protocol 1 One-sided Updatable Private Set Intersection Π_{uPSI}

Parameters: Threshold set size α.
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.
The protocol:

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 $|X^+| = |X^-| = |Y^+| = |Y^-| = \alpha$.
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 $I^{new} = (I^{old} \backslash I^-) \cup I^+$.

Figure 3: Our generic protocol for standard updatable private set intersection.

5.5. Correctness and Security

Theorem 1. *Protocol Π_{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. Lemma 1 ensures the correctness of the protocol, since the j-th execution of $\mathcal{F}_{\text{cPSI}}$ in protocol Π_{uPSI} computes the secret shares of the elements in the set I_j defined in the lemma. Regarding privacy of the protocol, 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 α using dummy inputs. Therefore, even the cardinalities do not leak.

When invoking $\mathcal{F}_{\text{CnP}}$, party P_1 obtains the sets I^- and I^+. It is important to note that, thanks to eq. (2) in Lemma 1, the outputs of these functionalities can indeed be interpreted as sets, since there are no repeated elements. We also note that the 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 only the input and output of P_1 in the ideal world. □

For the complete proof we refer to Appendix A.

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 $|X^{old}|$, $|Y^{old}|$.

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 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 $c^{new} = |X^{new} \cap Y^{new}|$ to P_1 only.

6.2. Protocol Description

Similar to the construction of Π_{uPSI} in Protocol 1, the protocol $\Pi_{\text{uPSI-card}}$ for cardinality in Protocol 2 executes four circuit-PSIs in 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 [33]).

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 α. By using the 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 substract 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

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

Parameters: Threshold set size α.
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.
The protocol:

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 $|X^+| = |X^-| = |Y^+| = |Y^-| = \alpha$.
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 $c^{new} = c^{old} + \mathsf{SS.Combine}(c^{(1)}, c^{(2)})$.

Figure 4: Our generic protocol for cardinality updatable private set intersection.

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 $n = |X| = |Y|$. The communication cost and the computation cost is computed as 4 times that of Π_{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 [7], we get an asymptotic complexity for communication cost of $O(\alpha \cdot \log(N/\alpha))$, and considering the proposal in [27], 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. $\square$

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 secret shares of the value v_x rather that 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.

7. 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.

7.1. 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, over which generic 2PC protocols [42]–[44] can be applied to evaluate arbitrary circuits.

In the particular case of updatable circuit PSI, where we propose that the old shared intersection is updated through four partial and smaller intersections, deleting elements becomes a point of contention. Indeed, the existing shares of the deleted elements should be deleted. As also noted in [11], 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 [11], authors argue that this situation results in only the leakage of the lifetime of the elements (i.e.

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.

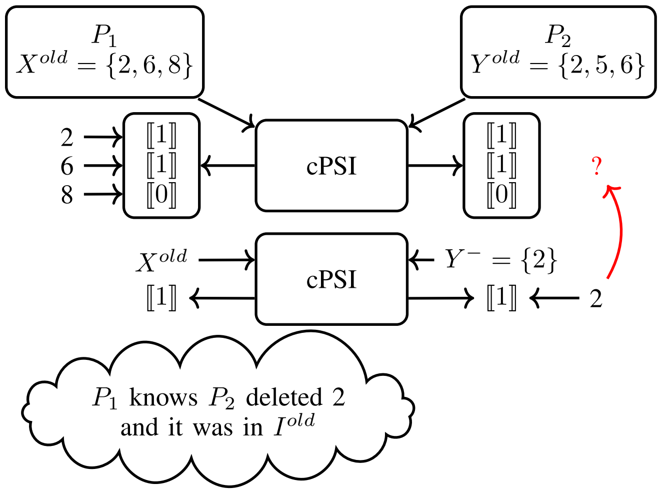


Figure 5: Inherent leakage in updatable circuit PSI.

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

Figure 5 uses the sets explained in Section 5.2, but the leakage is independent of the exact sets being used. As long as we are using unbalanced cPSI as a building block by applying it to small intersections, the deleted elements will need to be performed independently from the added ones, thus meaning that the situation explained below will appear. For ease of explanation, 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 partial 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 imbalance 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 element removed 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).

7.2. Work Arounds

We identify some work arounds to the challenge explained in the previous subsection.

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, while the cardinality circuit avoids the need for matching shares by aggregating everything with the addition (since the impact of every element is equivalent, the matching is not needed). 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 [11], 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). This situation comes up, for example, in the case of spam filters, where the blacklist is only increased by adding new spam mails and the mails being received in the inbox can be considered to only increase (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.

8. Performance Evaluation

8.1. Experimental setting

We have implemented[6] our constructions for updatable PSI and updatable cardinality PSI in Rust. Regarding the circuit PSI protocol Π_{cPSI}, we use the proposal by Karakoç and Küpçü [33], 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. To implement this scheme we have chosen to use the secure equality test by Ciampi and Orlandi in [45] with the `oblivious_transfer_protocols` library [46] for Oblivious Transfer extension and the `aes` library [47], as well as the Paillier encryption scheme [48] with the `kzen-paillier` library [49] 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 Π_{CnP}, we use the naïve approach described in [41, 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 that compares with our solution is [11][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 [11] 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 [11], we use the Linux `tc` command to simulate different network settings, with LAN at 0.2ms network latency and 1Gbps bandwidth, and WAN at 80ms network latency and testing several bandwidths of 200Mbps, 50Mbps, and 5Mbps.

The size of the datasets range from 2^{16} to 2^{17}. We had to stop at this size because the solution in [11] could not be executed with larger sizes threshold (the authors also mentioned that they had to increase their RAM for larger datasets). The size of updates (essentially α) vary from 2^4 to 2^6.

8.2. Implementation Optimization

We have made some optimizations to our protocol $\Pi_{\text{uPSI-card}}$ on our implementation due to the fact of using the specific unbalanced circuit PSI protocol from [33] as our building block Π_{cPSI}.

Let us first give a high level overview of the construction in [33]. 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 [33] as well as the secure equality test proposed in [45], 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.

8.3. Performance Results

We evaluate our protocols Π_{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. [11] supporting both addition and deletion (i.e., Section 4, in [11]). Table 2 depicts the obtained measurements. We have also evaluated the runtime and communication cost for sets bigger than those we were able to run the code from [11] on, and these results are shown in Table 3.

Communication cost. As expected, our solution for updatable cardinality PSI is slightly less expensive than our standard updatable PSI due to the additional use of the combine and permute protocol in our updatable PSI. Compared to [11], in general, our solution incurs in 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 [11]. Therefore, we outperform [11] as the update size grows.

It is worth noting that our solutions and [11] follow very different approaches leading to very different architectures. [11] uses an oblivious data structure for each dataset that is stored by the other party and can

6. Code will be released as open-source upon acceptance.

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

TABLE 2: Comparison of our constructions with state-of-the-art constructions from [11]. Best performances are highlighted in **bold**, and when our protocol outperforms [11], the corresponding cells are emphasized in green, otherwise, i.e., when [11] performs better, the actual cells are highlighted in blue.

Functionality	n	α	Protocol	Comm. (MB)	Online Run Time (s)			
					LAN	200 Mbps	50 Mbps	5 Mbps
PSI	2^{16}	2^4	[11]	**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	[11]	**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	[11]	**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	[11]	**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**

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

n	α	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.

α	Protocol	Comm. (MB)	Off-line Run Time (s)	Online Run Time (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

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, [11] 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 previously, adding communication cost.

With respect to the communication cost in bigger sets, we see that it becomes notably big. This is mostly due to the specific OT extension Rust library that is used in our implementation, which is not as highly optimized as some other C libraries as well as not implementing Silent OT extension [50].

Computation time. Similarly to the communication cost, as expected, the computation time is lower for the cardinality case than the standard PSI case, due to the lack of combine and permute execution. As for comparison results, our protocol outperforms [11], 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 [11]. Indeed, in our benchmarks, the computation time increases more significantly when the bandwidth is smaller (for example, $5Mbps$). Future work would be done to mitigate this through code optimizations.

Regarding computation time in the biggest case scenario, we see that our protocol with sets of 2^{20} elements even outperforms in the LAN setting that of [11] with sets of 2^{16} elements. The performance drops in the WAN settings, which is explainable due to the combination of the sensitivity of our implementation to bandwith and the high communication cost.

8.4. 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 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.

9. Conclusion

We have given new constructions for updatable PSI and updatable cardinality PSI, improving the current state-of-the-art [11] by $11\times$ to $40\times$ for updatable PSI, and by $14\times$ to $107\times$ for updatable cardinality PSI in comptutation time in the LAN setting.

Our work may inspire several avenues of follow-up works. One of these would be to continue the path toward 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 is how to commit the sets of each individual execution of the intersection without incurring in a linear communication and computation cost with respect to the big 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) are the execution of OT extension. To improve this concrete efficiency, a better OT extension library could be implemented or silent OT extension [50], [51] can be employed to have an even better asymptotic growth. Also with respect to the implementation, the code could be optimized to reduce the impact of low bandwidth on the protocol. Finally, the could could be further parallelized by using multiple threads per party, instead of only one.

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

References

[1] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert, "Mobile private contact discovery at scale," in *USENIX Security*, 2019.

[2] L. Hetz, T. Schneider, and C. Weinert, "Scaling mobile private contact discovery to billions of users," in *ESORICS*, 2023.

[3] N. Trieu, K. Shehata, P. Saxena, R. Shokri, and D. Song, "Epione: Lightweight contact tracing with strong privacy," 2020.

[4] A. Berke, M. Bakker, P. Vepakomma, K. Larson, and A. S. Pentland, "Assessing disease exposure risk with location data: A proposal for cryptographic preservation of privacy," 2020.

[5] M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, S. Saxena, K. Seth, M. Raykova, D. Shanahan, and M. Yung, "On deploying secure computing: Private intersection-sum-with-cardinality," in *EuroS&P*, 2020.

[6] S. Kamara, P. Mohassel, M. Raykova, and S. S. Sadeghian, "Scaling private set intersection to billion-element sets," in *FC*, N. Christin and R. Safavi-Naini, Eds., 2014.

[7] Y. Son and J. Jeong, "Psi with computation or circuit-psi for unbalanced sets from homomorphic encryption," in *ACM AsiaCCS*, 2023.

[8] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu, "Practical multi-party private set intersection from symmetric-key techniques," in *ACM CCS*, 2017.

[9] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *NDSS*, 2012.

[10] S. Badrinarayanan, P. Miao, and T. Xie, "Updatable private set intersection," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 2, pp. 378—-406, 2022.

[11] S. Badrinarayanan, P. Miao, X. Shi, M. Tromanhauser, and R. Zeng, "Updatable private set intersection revisited: Extended functionalities, deletion, and worst-case complexity," in *ASIACRYPT*, K.-M. Chung and Y. Sasaki, Eds., 2024.

[12] A. Agarwal, D. Cash, M. George, S. Kamara, T. Moataz, and J. Singh, "Updatable private set intersection from structured encryption," Cryptology ePrint Archive, Paper 2024/1183, 2024.

[13] G. Ling, P. Tang, and W. Qiu, "Efficient updatable PSI from asymmetric PSI and PSU," Cryptology ePrint Archive, Paper 2024/1712, 2024.

[14] R. Wang, J. Zhou, Z. Cao, X. Dong, and K.-K. Raymond Choo, "Updatable private set intersection with forward privacy," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 8573–8586, 2024.

[15] C. Meadows, "A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party," in *IEEE S&P*, 1986.

[16] B. Pinkas, T. Schneider, and M. Zohner, "Faster private set intersection based on ot extension," in *USENIX Security*, 2014.

[17] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious prf with applications to private set intersection," in *ACM CCS*, 2016.

[18] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "Spot-light: Lightweight private set intersection from sparse ot extension," in *CRYPTO*, 2019.

[19] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *EUROCRYPT*, 2004.

[20] C. Hazay, "Oblivious polynomial evaluation and secure set-intersection from algebraic prfs," in *TCC*, 2015.

[21] S. Ghosh and T. Nilges, "An algebraic approach to maliciously secure private set intersection," in *EUROCRYPT*, 2019.

[22] H. Chen, K. Laine, and P. Rindal, "Fast private set intersection from homomorphic encryption," in *ACM CCS*, 2017.

[23] H. Chen, Z. Huang, K. Laine, and P. Rindal, "Labeled psi from fully homomorphic encryption with malicious security," in *ACM CCS*, 2018.

[24] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "Psi from paxos: Fast, malicious private set intersection," in *EUROCRYPT*, 2020.

[25] P. Rindal and P. Schoppmann, "Vole-psi: Fast oprf and circuit-psi from vector-ole," in *EUROCRYPT*, 2021.

[26] S. Raghuraman and P. Rindal, "Blazing fast psi from improved okvs and subfield vole," in *ACM CCS*, 2022.

[27] M. Hao, W. Liu, L. Peng, H. Li, C. Zhang, H. Chen, and T. Zhang, "Unbalanced Circuit-PSI from oblivious Key-Value retrieval," in *USENIX Security*, 2024.

[28] R. A. Mahdavi, N. Lukas, F. Ebrahimianghazani, T. Humphries, B. Kacsmar, J. Premkumar, X. Li, S. Oya, E. Amjadian, and F. Kerschbaum, "Pepsi: practically efficient private set intersection in the unbalanced setting," in *USENIX Security*, 2024.

[29] F. Karakoç and A. Küpçü, "Linear complexity private set intersection for secure two-party protocols," in *CANS*, 2020.

[30] N. Chandran, D. Gupta, and A. Shah, "Circuit-psi with linear complexity via relaxed batch opprf," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 1, pp. 353—-372, 2022.

[31] Y. Yang, X. Liang, X. Song, Y. Dong, L. Huang, H. Ren, C. Dong, and J. Zhou, "Maliciously secure circuit private set intersection via spdz-compatible oblivious PRF," *Proceedings on Privacy Enhancing Technologies*, vol. 2025, no. 2, pp. 680–696, 2025.

[32] A. van Baarsen and M. Stevens, "Amortizing circuit-PSI in the multiple sender/receiver setting," *IACR Communications in Cryptology*, vol. 1, no. 3, 2024.

[33] F. Karakoç and A. Küpçü, "Enabling two-party secure computation on set intersection," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–12, 2025.

[34] Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas, "Private set intersection for unequal set sizes with mobile applications," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 4, pp. 177—-197, 2017.

[35] A. Abadi, C. Dong, S. J. Murdoch, and S. Terzis, "Multi-party updatable delegated private set intersection," in *FC*, 2022.

[36] G. Oded, *Foundations of Cryptography: Volume 2, Basic Applications*, 1st ed. USA: Cambridge University Press, 2009.

[37] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, Nov. 1979.

[38] M. Chase and P. Miao, "Private set intersection in the internet setting from lightweight oblivious prf," in *CRYPTO*, 2020.

[39] X. Song, D. Yin, J. Bai, C. Dong, and E.-C. Chang, "Secret-shared shuffle with malicious security," in *NDSS*, 2024.

[40] G. R. Chandran, T. Schneider, M. Stillger, and C. Weinert, "Concretely efficient private set union via circuit-based psi," in *ACM AsiaCCS*, 2025.

[41] M. Chase, E. Ghosh, and O. Poburinnaya, "Secret-shared shuffle," in *ASIACRYPT*, 2020.

[42] A. C.-C. Yao, "How to generate and exchange secrets," in *FOCS*, 1986.

[43] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *STOC*, 1987.

[44] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *STOC*, 1988.

[45] M. Ciampi and C. Orlandi, "Combining private set-intersection with secure two-party computation," in *SCN*, 2018.

[46] Harchandani, Lovesh, *Oblivious Transfer (OT), Oblivious Transfer Extensions (OTE) and multi-party protocols based on that*, 0th ed., 2025, https://crates.io/crates/oblivious_transfer_protocols.

[47] RustCrypto development team, *RustCrypto: Advanced Encryption Standard (AES)*, 0th ed., 2024, https://crates.io/crates/aes.

[48] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, 1999.

[49] G. Benattar, M. Cornejo, I. Leontiadis, M. Poumeyrol, O. Shlomovits, and D. Varlakov, *Paillier*, 0th ed., 2022, https://crates.io/crates/kzen-paillier.

[50] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl, "Efficient two-round ot extension and silent non-interactive secure computation," in *ACM CCS*, 2019.

[51] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient pseudorandom correlation generators: Silent ot extension and more," in *CRYPTO*, 2019.

[52] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *42nd FOCS*. IEEE Computer Society Press, Oct. 2001, pp. 136–145.

Appendix A. Security proofs

A.1. Security Definitions

In this section we give the formal definitions for the security notions we will be using for the proofs.

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
$$\Pr\left[\mathsf{SS.Combine}\left(s^{(X)}, s^{(Y)}\right) \neq x\right] = \mathsf{negl}(\kappa),$$
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* Π *be a 2PC protocol, let* $\mathit{View}_i^\Pi(X;Y)$ *be the view of party* P_i *(including input, random tape and all received messages) of the protocol* Π *with inputs* X *by* P_X *and* Y *by* P_Y*, and let* $\mathit{Out}_i^\Pi(X;Y)$ *be the output of* P_i *in protocol* Π *on inputs* X *by* P_X *and* Y *by* P_Y*. Then,* Π *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*

$$\begin{gathered}\mathit{View}_X^\Pi(X;Y), \mathit{Out}_X^\Pi(X;Y)\\ \approx_c\\ \mathit{Sim}_X(x, \mathcal{F}_X(X;Y)), \mathcal{F}_Y(X;Y),\\ \mathit{View}_Y^\Pi(X;Y), \mathit{Out}_Y^\Pi(X;Y)\\ \approx_c\\ \mathit{Sim}_Y(x, \mathcal{F}_Y(X;Y)), \mathcal{F}_X(X;Y),\end{gathered}$$

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

The definition above applies also to protocols described in hybrid worlds. We recap below the notion of protocol composition.

We consider the following theorem.

Theorem 3 (Composition Theorem)**.** *Let* Π_F *be a protocol realizing* $\mathcal{F}_F$ *in the* $\mathcal{F}_G$*-hybrid model, and let* Π_G *be a* *protocol realizing $\mathcal{F}_G$ in the $\mathcal{F}_H$-hybrid model, Define the composed protocol*

$$\Pi'_F = \Pi_F^{\mathcal{F}_G \rightarrow \Pi_G},$$

by replacing each invocation of $\mathcal{F}_G$ in Π_F with a real instantiation of Π_G. Then Π'_F realizes $\mathcal{F}_F$ in the $\mathcal{F}_H$-hybrid model. Moreover, Π'_F realizes $\mathcal{F}$ against semi-honest adversaries even under parallel composition. *Here, parallel composition means running multiple instances of the same or different protocols at the same time, with messages possibly interleaved.*

Notice that definition 3 refers to the so-called stand-alone security model. Thus, universal composability is not guaranteed in the malicious setting [52]. However, the simpler notion of stand-alone security against semi-honest adversaries is preserved under parallel composition.

Proof Sketch. We define the simulator for Π'_{F} to be the simulator that run first the simulator $\mathcal{S}_{\mathrm{F}}$ of Π_{F}, subsequentely, it iterates over all the simulated call to $\mathcal{F}_{\mathrm{G}}$ created by the simulator $\mathcal{S}_{\mathrm{F}}$, and run $\mathcal{S}_{\mathrm{G}}$ on the input and outputs generated by $\mathcal{S}_{\mathrm{F}}$. Notice, that for parallel composition we can re-order the messages generated by $\mathcal{S}_{\mathrm{G}}$ after the above process to generate a simulated view according to the parallel execution of the real protocol Π'_{F}.

We prove indistinguishability using a hybrid argument. First, we replace all calls to $\mathcal{S}_G$ with real executions of protocol Π_G. By definition 3, this hybrid experiment is indistinguishable from the simulated world. Finally, we replace the single call to the simulator $\mathcal{S}_F$ with a real execution of protocol Π_F. Again, by definition 3, the resulting hybrid experiment is indistinguishable from the real world. □

A.2. Proofs

Lemma 2. *Let $X^{old}, X^+, X^-, Y^{old}, Y^+, Y^-$ be sets and X^+, X^- (resp. Y^+, Y^-) are valid updates for X^{old} (resp. Y^{old}) and let $I^{old} = X^{old} \cap Y^{old}$. Set*

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

Then

$$X^{new} \cap Y^{new} = (I^{old} \setminus (I_3 \cup I_4)) \cup (I_1 \cup I_2) \tag{3}$$
$$I_1 \cap I_2 = I_3 \cap I_4 = \emptyset \tag{4}$$

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

For ease of comprehension of the proof we refer to Figure 2.

Proof. First we get that :

$$\begin{aligned} I^{new} &= (X^{new} \cap (Y^{old} \setminus Y^-)) \cup (X^{new} \cap Y^+) && (5)\\ &= ((X^{old} \setminus X^-) \cap (Y^{old} \setminus Y^-)) \cup (X^+ \cap (Y^{old} \setminus Y^-)) \\ &\cup (X^{new} \cap Y^+), && (6) \end{aligned}$$

where at steps 5, 6 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

$$\begin{aligned} ((X^{old} \setminus X^-) \cap (Y^{old} \setminus Y^-)) =& (X^{old} \setminus (X^- \cup Y^-)) \\ & \cap (Y^{old} \setminus (X^- \cup Y^-), \end{aligned}$$

in fact, any element in $X^{old} \setminus X^-$ but not in $X^{old} \setminus (X^- \cup Y^-)$ is then in $X^{old} \cap Y^-$ (and in particular in Y^-) and therefore not in the intersection $(X^{old} \setminus X^-) \cap (Y^{old} \setminus Y^-)$ (and analogously for $Y^{old} \setminus Y^-$). Subbing the equation above into eq. (6) and setting $I_1 := X^{new} \cap I^+$ we have:

$$\begin{aligned} I^{new} &= \big((X^{old} \setminus (X^- \cup Y^-)) \cap (Y^{old} \setminus (X^- \cup Y^-))\big) \\ &\cup (X^+ \cap (Y^{old} \setminus Y^-)) \cup I_1 \\ &= (I^{old} \setminus (X^- \cup Y^-)) \cup (X^+ \cap (Y^{old} \setminus Y^-)) \cup I_1 && (7) \\ &= I^{old} \setminus ((X^- \cup Y^-) \cap (X^{old} \cap Y^{old})) \\ &\cup (X^+ \cap (Y^{old} \setminus Y^-)) \cup I_1 && (8) \end{aligned}$$

At step 7 we have applied the right distributive property of set difference over set intersection and $I^{old} = X^{old} \cap Y^{old}$, at step 8 we have used the definition of set difference. For convenience, we set $I_2 := (X^+ \cap (Y^{old} \setminus Y^-))$ and $I^- := ((X^- \cup Y^-) \cap (X^{old} \cap Y^{old}))$. Thus

$$I^{new} = I^{old} \setminus I^- \cup (I_2 \cup I_1) \tag{9}$$

We focus now on I^-:

$$\begin{aligned} I^- &= (X^- \cap (X^{old} \cap Y^{old})) \cup (Y^- \cap (X^{old} \cap Y^{old})) && (10) \\ &= (X^- \cap Y^{old}) \cup (Y^- \cap X^{old}) && (11) \\ &= (X^- \cap (Y^{old} \setminus Y^-)) \cup (Y^- \cap X^{old}) && (12) \end{aligned}$$

Where at step 10 we have applied the distributive property between set intersection and set union, at step 11 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 12 we have used that any element in $X^- \cap Y^{old}$ not in $X^- \cap (Y^{old} \setminus X^-)$ must be in $Y^- \cap X^- \subseteq Y^{old} \cap Y^-$ since $Y^- \subseteq Y^{old}$. Setting $I_3 := (X^{old} \cap Y^-)$ and $I_4 := (X^- \cap (Y^{old} \setminus Y^-))$ and plugging the above derivation for I^- in Equation (9), we obtain Equation (3) in the statement of the lemma.

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

Theorem 4. *Protocol Π_{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. We first show that the protocol is correct. First we notice that the four call to $\mathcal{F}_{\mathrm{cPSI}}$ 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 $\mathcal{F}_{\mathrm{CnP}}$ can be indeed parsed as a set. Thus, by Equation (1) 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 $\vec{0}$ of the size α, 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 (2) it follows that $\bar{I}^{-} = I^{-}$ and $\bar{I}^{+} = I^{+}$. $\square$