Updatable Private Set Intersection

Saikrishna Badrinarayanan * Peihan Miao † Tiancheng Xie ‡

Abstract

Private set intersection (PSI) allows two mutually distrusting parties each with a set as input, to learn the intersection of both their sets without revealing anything more about their respective input sets. Traditionally, PSI studies the *static* setting where the computation is performed only once on both parties' input sets.

We initiate the study of updatable private set intersection (UPSI), which allows parties to compute the intersection of their private sets on a regular basis with sets that also constantly get updated. We consider two specific settings. In the first setting called *UPSI with addition*, parties can add new elements to their old sets. We construct two protocols in this setting, one allowing both parties to learn the output and the other only allowing one party to learn the output. In the second setting called *UPSI with weak deletion*, parties can additionally delete their old elements every t days. We present a protocol for this setting allowing both parties to learn the output. All our protocols are secure against semi-honest adversaries and have the guarantee that both the computational and communication complexity only grow with the set updates instead of the entire sets.

Finally, we implement our UPSI with addition protocols and compare with the state-of-the-art PSI protocols. Our protocols compare favorably when the total set size is sufficiently large, the new updates are sufficiently small, or in networks with low bandwidth.

*Visa Research. Email: sabadrin@visa.com

†University of Illinois at Chicago. Email: peihan@uic.edu

‡University of California, Berkeley. Email: tianc.x@berkeley.edu

Contents

1 Introduction

Private set intersection (PSI) enables two parties, each holding a private set of elements, to compute the intersection of the two sets while revealing nothing more than the intersection itself. Over the years, PSI and its related functionalities have found many real-world privacy-preserving applications including DNA testing and pattern matching [TPKC07], remote diagnostics [BPSW07], online advertising [IKN+20], password breach alerting [TPY+19], mobile private contact discovery [KRS+19], privacy-preserving contact tracing [TSS+20,CCF+20], and many more. There has been tremendous progress made towards realizing PSI efficiently [KKRT16,RR17,CLR17,PRTY19,CM20,PRTY20] with both semi-honest and malicious security.

Despite tremendous advancements and improvements in the efficiency of PSI protocols, one drawback of all the existing protocols is that when parties update their sets to include some new elements or remove certain existing elements, in order to compute the intersection between the two updated sets, parties have to perform a fresh PSI computation every time. This incurs a lot of wasteful computational and communication overhead, especially in scenarios where the updates are done very frequently and/or the updates to the existing sets are small. Indeed, in a lot of real-world scenarios such as aggregated ads measurement [IKN+20], password breach monitoring [APP,MIC], digital contact tracing [TSS+20, CCF+20], PSI is performed on a regular (e.g., daily) basis with *updated* sets, where the daily update to the sets could be very small compared to the entire sets. In this work, we ask the following question:

Can we design protocols that allow parties to regularly update their sets and perform PSI where every time both the computation and communication costs are only proportional to their updates instead of the entire sets?

1.1 Our Results

We first formalize the notion of *updatable private set intersection (UPSI)* as a special case of secure two-party computation with a reactive functionality that interacts with both parties over many days and keeps its own private internal state between days. There are two types of updates to consider: adding new elements and deleting existing elements. In particular, we consider the following two settings and present three constructions summarized in Table 1.

UPSI with Addition. In the first setting, on every day, we allow both parties to *add* a new set of elements to their existing old sets. The output on each day is the intersection of the two updated entire sets. We construct two protocols:

- *Two-Sided UPSI with Addition:* A Diffie-Hellman based protocol that allows *both* parties to receive the output on each day. Both the computational and communication complexity of this protocol only grow linearly with the size of the added new sets and are independent of the size of the old sets.
- *One-Sided UPSI with Addition:* An additively homomorphic encryption based protocol that allows only *one* party to receive the output. The overall complexity may vary on different days, hence we consider the amortized cost per day over a long period of days. Both the *amortized* computational and communication complexity of this protocol only grow linearly with the size of the added new sets and logarithmically with the size of the old sets. Technically, we develop an ORAM-like tree structure that allows one party to obliviously update an encrypted

database and another party to obliviously search on the encrypted database (where the secret key is held by the first party), which may be of independent interest.

Note that one-sided UPSI with addition is a stronger functionality in the semi-honest setting because the output-receiving party can send the output to the other party so as to achieve two-sided output. We present a protocol for one-sided UPSI with addition because the functionality may be desirable in many server-client applications where only the client is allowed to learn the output (e.g., password breach monitoring [APP, MIC]).

UPSI with Weak Deletion. In the second setting, we additionally allow both parties to refresh their sets every t days. Namely, they will add a set of elements to their sets every day, and delete elements that were added to their sets t days ago. This setting is motivated by applications such as privacy-preserving contact tracing [TSS+20, CCF+20] where data about people's interactions from more than e.g. 14 days ago is no longer useful. In this example, one party's (server's) input is the set of people who tested positive on that day, the other party's (client's) input is the set of people they interacted with on that day. The output on each day is the list of people the client interacted with in the last t days, who also tested positive in the last t days.

We construct an oblivious transfer (OT) based protocol that allows both parties to receive the output. Both the computational and communication complexity grow linearly with the size of the added new sets and t.

Functionality	Output	Protocol	Comp. Complexity	Comm. Complexity
Addition-Only	Two-Sided	Figure 4	$O(N')$	$O(N')$
Addition-Only	One-Sided	Figure 6	$O^*(N' \log N)$	$O^*(N' \log N)$
Weak Deletion	Two-Sided	Figure 9	$O(N' \cdot t)$	$O(N' \cdot t)$

Table 1: Summary of our protocols. N denotes the size of the old sets and N' denotes the size of the updates. t denotes the number of days when parties refresh their sets in UPSI with weak deletion. $O^*(\cdot)$ denotes amortized complexity.

Experiments. We implement the two UPSI with addition protocols and compare with the state-of-the-art PSI protocols. To demonstrate the updatable property, we consider the following setting: each party initially holds an empty set. Then, on every new day, both parties add a new set of size N' to their existing sets and wish to learn the updated set intersection. We repeat this process over a period of several days ($\frac{N}{N'}$) till the total set size of each party is N. We compare the amortized (over the total number of days) communication cost and running time of our protocol with the prior PSI protocols [KKRT16, PRTY19, CM20], where, on any day, the two parties run a fresh PSI on their updated sets to learn the updated intersection.

Generally speaking, the (concrete/amortized) communication cost of both our protocols only grows with N' and at most logarithmically with N, hence we have more advantages in efficiency when the total set size N is larger, the update size N' is smaller, and the network bandwidth is lower. In particular, our two-sided UPSI with addition protocol beats all the PSI protocols in communication by $7.5 - 13250\times$ in the settings we consider (where $N \gg N'$). As an example for running time, when $N = 2^{20}$ and $N' = 2^{10}$, our protocol beats the best PSI protocol by $1.1 - 7.6\times$ for network bandwidth between $5 - 50$ Mbps. Our one-sided UPSI with addition protocol beats the PSI protocols in communication by $2 - 149\times$ in almost all settings we consider. As an example for

running time, when $N = 2^{20}$ and $N' = 2^6$, our protocol beats the best PSI protocol by $1.8 - 30.5\times$ for network bandwidth between $5 - 50$ Mbps.

1.2 Related Work

There are various approaches in achieving efficient semi-honest PSI in different settings, including Diffie-Hellman-based [Mea86, HFH99], fully homomorphic encryption (FHE)-based [CLR17], circuit-based [HEK12, PSSZ15, PSWW18, PSTY19], and oblivious transfer (OT)-based [KKRT16, PRTY19, CM20] protocols. We refer the reader to [PSZ14, PSZ18] for an overview of the different paradigms for PSI. Protocols based on OT [KKRT16, PRTY19, CM20] are currently the fastest in practice because they can take advantage of the efficient implementation of OT extension [IKNP03, ALSZ13].

In the updatable setting, the work of Kiss et al. [KLS+17] studies PSI with pre-computation between a server with a large set of size N and a client with a small set of size N'. In a setup phase, the communication and computation cost is linear in N while in the online phase the cost is only linear in N'. It allows the server to update its set without recomputing the setup phase and the client to run the online phase for new sets. Nevertheless, they do not provide an ideal functionality for the updatable setting that captures the exact leakage from their protocols. In particular, if the client's sets in the online phase are $X_1, \ldots, X_d$ and the server's updates are $Y_1, \ldots, Y_d$, then all of their protocols reveal to the client $X_i \cap Y_j$ for all i, j. Such leakage also arises in our attempt to extend the Diffie-Hellman-based PSI to the updatable setting, which we discuss in Section 1.3. In this work, we formalize security by a reactive ideal functionality that prevents such leakage in the updatable setting. Buddhavarapu et al. [BKM+20] consider the updatable setting where only one party's dataset is updated and arrives in a streaming fashion, and the output intersection is additively secret shared amongst both parties. Furthermore, they require an upper bound on this streaming input size to be known apriori. In our case, both parties' sets can be updated with no apriori upper bound needed.

A recent work of Abadi et al. [ATD20] studies delegated PSI protocols that support data updates and multi-party PSI. In particular, clients can upload their (encrypted) private data to a server and outsource the PSI computation. Clients can update their sets with communication and computation only growing with their updates. However, both the computation and communication of the PSI protocol grow with the entire sets, and they require the existence of a server.

1.3 Challenges and Ideas

We briefly explain the technical challenges in the design of our protocols. We start with the addition-only setting. Let X, Y denote the old sets of the two parties P_0, P_1 respectively, and let X', Y' denote their new added sets. For simplicity, assume $|X| = |Y| = N$ and $|X'| = |Y'| = N'$. Recall that we are mostly interested in the scenario when $N \gg N'$ and our goal is to make the computation and communication cost to learn the new intersection only grow with N' and not N (except with logarithmic factors).

First, note that naturally extending existing FHE-based [CLR17], circuit-based [HEK12, PSSZ15, PSWW18, PSTY19], or OT-based [KKRT16, PRTY19, CM20] PSI protocols does not work. In the FHE-based protocols, while P_0 (the output-receiving party) can send $\mathsf{Enc}(X')$ which only grows with N', the computation cost of P_1 would involve homomorphically evaluating to compare with his entire input set $Y \cup Y'$ (and also homomorphically compare Y' with P_0's old set X), which

grows with N. A similar issue arises in circuit-based protocols where in fact, communication also grows with N. The OT-based protocols require one party to fix its input set and the number of OTs (to set up the oblivious pseudorandom function) depends on N, so both communication and computation would grow.

Two-sided UPSI. On first thought, the Diffie-Hellman-based protocol [Mea86, HFH99] seems more promising because it has special algebraic structures that may be suitable for the updatable setting. To briefly recall the protocol, let X, Y be P_0 and P_1's input sets, respectively. Both parties first hash their elements into a group where DDH holds, namely $H(X)$ and $H(Y)$. Each party picks a secret exponentiation key, that is k_0 and k_1 respectively. P_1 then sends $H(Y)^{k_1}$ and P_0 responds back with $H(Y)^{k_0 k_1}$. Symmetrically, they can obtain $H(X)^{k_0 k_1}$. By comparing $H(Y)^{k_0 k_1}$ and $H(X)^{k_0 k_1}$, both parties can compute the intersection $X \cap Y$. In the updatable setting, they can repeat this process on their new elements X', Y' ensuring that computation and communication only grow with the size of the new sets. Unfortunately, this naïve adaption to the updatable setting does not trivially solve the problem as it leaks extra information than what the parties can learn from the ideal functionality. In particular, it leaks $X' \cap Y$ and $X' \cap Y'$ to P_0, which is not available in the ideal world.

Our solution is to get rid of such leakage by investigating what can be inferred from the ideal functionality and leveraging the nice algebraic structures. In particular, we split the updated output into two parts, one of which (that is, $X \cap Y'$) can be computed by extending the above DDH-based protocol and for the other (in particular, $X' \cap (Y \cup Y')$), we run a fresh PSI instance on small input sizes. We carefully choose this split and design the appropriate sub-protocols to ensure no information is leaked. We refer to Section 4 for a detailed overview and the formal construction.

One-sided UPSI. In our protocol above, we crucially rely on the fact that both parties learn the output on each day. In particular, even if we want only P_0 to learn output, to ensure that P_1 uses a small input for the fresh PSI, we require P_1 to learn the output of the first part that extends the DDH-based approach. We now focus on the challenges and ideas in designing a protocol for one-sided UPSI where only P_0 learns the output.

At a high level, our key idea is for P_1 to store an encrypted version of his set on P_0's storage and on each day, he updates this encrypted database based only on his new input Y'. Then, we require a mechanism that allows P_0 to obliviously query this database and compute on the encrypted data (by interacting with P_1) to learn the intersection without leaking any information to P_1.

We discuss one natural idea to implement this mechanism using FHE. Suppose P_1 uses FHE to encrypt Y and stores $\mathsf{Enc}(Y)$ on P_0. Then P_0 can use her inputs to homomorphically compute $\mathsf{Enc}(X \cap Y)$. Both parties can then run a secure two-party computation (2PC) protocol where P_0's input is $\mathsf{Enc}(X \cap Y)$ and P_1's input is secret key sk, from which P_0 learns the output $(X \cap Y)$. When there is update, P_1 can update the encrypted database by sending $\mathsf{Enc}(Y')$ and P_0 can learn $(X' \cap (Y \cup Y'))$ with communication only growing with N'. However, P_0's homomorphic computation still grows with N. Moreover, it requires expensive FHE evaluation and 2PC for FHE decryption.

To implement this approach efficiently, we take inspiration from oblivious RAM [SvDS+18]. The crucial idea is that the encrypted database is maintained in a tree structure where, on any day, P_1 only updates one level of the tree and P_0 only queries on one path of the tree, so the (amortized) cost only grows with the depth of the tree (logarithmic in N and not linear). We also build an

efficient 2PC protocol for decryption using additively homomorphic encryption instead of FHE. We further optimize our protocol by using Cuckoo hashing [PR04] to store elements in each node of the tree and leveraging the structure of El Gamal encryption [Gam84] in our context. We refer to Section 5 for more details.

Weak Deletion. We make an interesting observation about OT-based PSI protocols [KKRT16, PRTY19, CM20]. They work in a *streaming* setting where, in a setup phase, only the output-receiving party's input set is known. Then, the sender's inputs can be fed in a streaming manner and the protocol allows the receiver to learn the intersection for each stream. We directly take advantage of this streaming structure and build on these protocols to design our weak deletion protocol. We refer to Section 6 for an overview and the construction.

2 Preliminaries

Notation. We use λ, σ to denote the computational and statistical security parameters, respectively. By $\mathsf{negl}(\lambda)$ we denote a negligible function, i.e., a function f such that $f(\lambda) < 1/p(\lambda)$ holds for any polynomial $p(\cdot)$ and sufficiently large λ. By $\overset{c}{\approx}$ we mean two distributions are computationally indistinguishable. Let $\mathbb{N}^+$ denote the list of positive integers and $\mathbb{N}$ denote $\mathbb{N}^+ \cup \{0\}$.

2.1 Private Set Intersection (PSI)

Private Set Intersection (PSI) is a special case of secure two-party computation. We follow the standard security definition for semi-honest secure two-party computation. Consider two parties P_0, P_1 with input sets X, Y of size N_0, N_1, respectively. Their goal is to run a two party secure computation protocol Π at the end of which party P_0 learns the set intersection $I = X \cap Y$.[1] The formal definition of the ideal functionality is shown in Figure 1.

Parameters: The set size of X is N_0 and the set of Y is N_1.

Inputs:
Party P_0 has an input set X of size N_0 where each element is from $\{0,1\}^*$.
Party P_1 has an input set Y of size N_1 where each element is from $\{0,1\}^*$.

Output: P_0 receives the set intersection $I = X \cap Y$.

Figure 1: Ideal functionality $\mathcal{F}_{\mathsf{PSI}}$ for two-party PSI.

Let $\mathsf{View}_0^{\Pi}(X,Y)$ and $\mathsf{View}_1^{\Pi}(X,Y)$ be the view of P_0 and P_1 in the protocol Π, respectively. Let $\mathsf{Out}^{\Pi}(X,Y)$ be the output of P_0 in the protocol. Let $f(X,Y)$ be the output of P_0 in the ideal functionality. The protocol Π is semi-honest secure if there exists PPT simulators Sim_0 and Sim_1 such that for all inputs X, Y,

$$\mathsf{View}_0^{\Pi}(X,Y) \overset{c}{\approx} \mathsf{Sim}_0\left(1^\lambda, X, N_1, f(X,Y)\right),$$
$$\left(\mathsf{View}_1^{\Pi}(X,Y), \mathsf{Out}^{\Pi}(X,Y)\right) \overset{c}{\approx} \left(\mathsf{Sim}_1(1^\lambda, Y, N_0), f(X,Y)\right).$$

[1] Another formulation is allowing both parties to learn the output, which can be easily achieved in the semi-honest model by P_0 sending the output I to P_1 at the end.

2.2 Tools and Assumptions

Additively Homomorphic Encryption. An additively homomorphic encryption scheme is a public-key encryption scheme $\mathsf{AHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ over message space $\mathcal{M}$ with correctness, CPA security, and linear homomorphism.

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
- $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m; r)$
- $m/\perp \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$
- Homomorphic addition: $\mathsf{Enc}_{\mathsf{pk}}(m_1) \oplus \mathsf{Enc}_{\mathsf{pk}}(m_2) = \mathsf{Enc}_{\mathsf{pk}}(m_1 + m_2)$ for $\forall m_1, m_2 \in \mathcal{M}$.
- Homomorphic multiplication with constant: $c \odot \mathsf{Enc}_{\mathsf{pk}}(m) = \mathsf{Enc}_{\mathsf{pk}}(c \cdot m)$ for $\forall c, m \in \mathcal{M}$.

We implicitly assume that each homomorphic evaluation is followed by a refresh operation, where the resulting ciphertext is added with an independently generated encryption of zero. This is required in our protocols to ensure that the randomness of the final ciphertext is independent of the randomness used in the original set of ciphertexts. For the popular additively homomorphic encryption schemes such as ElGamal encryption [Gam84] (based on DDH) and Paillier encryption [Pai99] (based on the Decisional Composite Residuosity assumption), a homomorphically evaluated ciphertext is statistically identical to a fresh ciphertext. We refer to [Gam84, Pai99] for formal definitions of correctness and CPA security.

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

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

Cuckoo Hashing. We define Cuckoo Hashing [PR04] verbatim from [KKRT16]. To assign n items into b bins, first choose random functions $H_1, H_2, H_3 : \{0,1\}^* \to [b]$ and initialize empty bins $B[1, \ldots, b]$. To hash an item x, first check to see whether any of the bins $B[H_1(x)], B[H_2(x)], B[H_3(x)]$ are empty. If so, place x in one of the empty bins and terminate. Otherwise, choose a random $i \in \{1, 2, 3\}$, evict the item currently in $B[H_i(x)]$, replacing it with x, and then recursively try to insert the evicted item. If this process does not terminate after a certain number of iterations, then the final evicted element is placed in a special bin called stash.

3 Updatable Private Set Intersection

In this section, we formalize the definition of *Updatable Private Set Intersection (UPSI)*. Consider two parties P_0 and P_1 who wish to run PSI on a daily basis with updated sets each day. We consider two settings on how they can update their sets. The first setting, which we call *UPSI with addition*, allows both parties to add a set of elements to their respective sets each day. In the second setting, which we call *UPSI with weak deletion*, both parties can add a set of elements to their sets every day and delete elements that were added to their set t days before. In other words, each party only holds the elements added in the most recent t days. Moreover, on each day, the output learnt is only the intersection of each party's new elements with the last t days' elements of the other party.

3.1 UPSI with Addition

In the setting of UPSI with addition, two parties P_0 and P_1 each hold a private set and add new elements to their respective sets each day. They want to jointly compute their set intersection every day on their updated sets without revealing anything beyond that. We formalize *UPSI with addition* as a special case of secure two-party computation with a reactive functionality defined in Figure 2. For simplicity, we assume that each party adds the same number of elements as the other party on each day.

We consider two output scenarios: in *two-sided* UPSI with addition $\mathcal{F}_{\mathsf{UPSI\text{-}add\text{-}two}}$, both parties obtain output at the end of each day; in *one-sided* UPSI with addition $\mathcal{F}_{\mathsf{UPSI\text{-}add\text{-}one}}$, only P_0 gets the output. Note that in the semi-honest model, a secure protocol achieving $\mathcal{F}_{\mathsf{UPSI\text{-}add\text{-}one}}$ can be easily transformed into one achieving $\mathcal{F}_{\mathsf{UPSI\text{-}add\text{-}two}}$ by P_0 sending the output to P_1 at the end, hence $\mathcal{F}_{\mathsf{UPSI\text{-}add\text{-}one}}$ is a stronger notion in the semi-honest model.

Initialization: $X := \emptyset, Y := \emptyset$.

Day d**:**

- **Public parameter:** The set size on Day d is N_d.
- **Inputs:**
  P_0 inputs a set X_d of size N_d where each element is from $\{0,1\}^*$, and $X_d \cap X = \emptyset$.
  P_1 inputs a set Y_d of size N_d where each element is from $\{0,1\}^*$, and $Y_d \cap Y = \emptyset$.
- **Update:** On receiving the inputs from the two parties, the ideal functionality updates $X := X \cup X_d, Y := Y \cup Y_d$ and computes $I_d = X \cap Y$.
- **Output:**
  In $\mathcal{F}_{\mathsf{UPSI\text{-}add\text{-}two}}$, the ideal functionality sends I_d to both parties.
  In $\mathcal{F}_{\mathsf{UPSI\text{-}add\text{-}one}}$, the ideal functionality sends I_d to only P_0.

Figure 2: Ideal functionalities $\mathcal{F}_{\mathsf{UPSI\text{-}add\text{-}two}}$ and $\mathcal{F}_{\mathsf{UPSI\text{-}add\text{-}one}}$ for UPSI with addition.

Consider the first D days: let $X_{[D]} = \{X_1, \ldots, X_D\}$ be the inputs of P_0 and $Y_{[D]} = \{Y_1, \ldots, Y_D\}$ be the inputs of P_1. Let $\mathsf{View}_b^{\Pi,D}(X_{[D]}, Y_{[D]})$ and $\mathsf{Out}_b^{\Pi,D}(X_{[D]}, Y_{[D]})$ be the view and outputs of P_b ($b \in \{0,1\}$) in the protocol Π at the end of D days, respectively. Let $f(X_{[D]}, Y_{[D]}) := \{I_1, \ldots, I_D\}$ be the outputs of the ideal functionality in the D days.

Definition 3.1. *(**Two-Sided UPSI with Addition.**) A protocol Π is semi-honest secure with respect to ideal functionality $\mathcal{F}_{\mathsf{UPSI\text{-}add\text{-}two}}$ if there exists PPT simulators Sim_0 and Sim_1 such that for any $D \in \mathbb{N}^+$, any inputs $(X_{[D]}, Y_{[D]})$,*

$$\left(\mathsf{View}_0^{\Pi,D}(X_{[D]}, Y_{[D]}), \mathsf{Out}_1^{\Pi,D}(X_{[D]}, Y_{[D]})\right) \overset{c}{\approx} \left(\mathsf{Sim}_0\left(1^\lambda, X_{[D]}, f(X_{[D]}, Y_{[D]})\right), f(X_{[D]}, Y_{[D]})\right),$$
$$\left(\mathsf{View}_1^{\Pi,D}(X_{[D]}, Y_{[D]}), \mathsf{Out}_0^{\Pi,D}(X_{[D]}, Y_{[D]})\right) \overset{c}{\approx} \left(\mathsf{Sim}_1\left(1^\lambda, Y_{[D]}, f(X_{[D]}, Y_{[D]})\right), f(X_{[D]}, Y_{[D]})\right).$$

Definition 3.2. *(**One-Sided UPSI with Addition.**) A protocol Π is semi-honest secure with respect to ideal functionality $\mathcal{F}_{\mathsf{UPSI\text{-}add\text{-}one}}$ if there exists PPT simulators Sim_0 and Sim_1 such that for any $D \in \mathbb{N}^+$, any inputs $(X_{[D]}, Y_{[D]})$,*

$$\mathsf{View}_0^{\Pi,D}(X_{[D]}, Y_{[D]}) \overset{c}{\approx} \mathsf{Sim}_0\left(1^\lambda, X_{[D]}, f(X_{[D]}, Y_{[D]})\right),$$

$$\left(\mathsf{View}_1^{\Pi,D}(X_{[D]}, Y_{[D]}), \mathsf{Out}_0^{\Pi,D}(X_{[D]}, Y_{[D]})\right) \stackrel{c}{\approx} \left(\mathsf{Sim}_1\left(1^\lambda, Y_{[D]}\right), f(X_{[D]}, Y_{[D]})\right).$$

3.2 UPSI with Weak Deletion

In the setting of UPSI with weak deletion, two parties P_0 and P_1 each hold a private set. Then, on each day, they add new elements to their respective sets and delete elements that were added t days before. On each day, they want to jointly compute the union of the intersection between their new elements and the other party's updated set comprising elements from the last t days, without revealing anything beyond that. We formalize *UPSI with weak deletion* as a special case of secure two-party computation with a reactive functionality defined in Figure 3. For simplicity, we assume that each party adds the same number of elements as the other party on each day. We only consider two-sided output where both parties receives the output every day.

Initialization: $X := \emptyset, Y := \emptyset$.

Day d**:**

- **Public parameter:** The set size on Day d is N_d.
- **Inputs:**
  P_0 inputs a set X_d of size N_d where each element is from $\{0,1\}^*$, and $X_d \cap X = \emptyset$.
  P_1 inputs a set Y_d of size N_d where each element is from $\{0,1\}^*$, and $Y_d \cap Y = \emptyset$.
- **Update:** On receiving inputs from both parties, the ideal functionality updates $X := (X \cup X_d) \setminus X_{d-t}, Y := (Y \cup Y_d) \setminus Y_{d-t}$ and computes $I_d = \left((X_d \cap Y) \cup (X \cap Y_d)\right)$. (If $d - t \leq 0$, let $X_{d-t} = Y_{d-t} = \emptyset$.)
- **Output:** The ideal functionality sends I_d to both parties.

Figure 3: Ideal functionality $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$ for UPSI with weak deletion.

Consider the first D days: let $X_{[d]} = \{X_1, \ldots, X_D\}$ be the inputs of P_0, $Y_{[D]} = \{Y_1, \ldots, Y_D\}$ be the inputs of P_1 and $N_{[D]} = \{N_1, \ldots, N_D\}$ be the set sizes. Let $\mathsf{View}_b^{\Pi,D}(X_{[D]}, Y_{[D]})$ and $\mathsf{Out}_b^{\Pi,D}(X_{[D]}, Y_{[D]})$ be the view and outputs of P_b ($b \in \{0,1\}$) in the protocol Π at the end of D days, respectively. Let $f(X_{[D]}, Y_{[D]}) := \{I_1, \ldots, I_D\}$ be the outputs of the ideal functionality in D days.

Definition 3.3. ***(UPSI with Weak Deletion.)*** *A protocol* Π *is semi-honest secure with respect to ideal functionality* $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$ *if there exists PPT simulators* Sim_0 *and* Sim_1 *such that for any* $d \in \mathbb{N}^+$*, any inputs* $(X_{[D]}, Y_{[D]})$*,*

$$\left(\mathsf{View}_0^{\Pi,D}(X_{[D]}, Y_{[D]}), \mathsf{Out}_1^{\Pi,D}(X_{[D]}, Y_{[D]})\right) \stackrel{c}{\approx} \left(\mathsf{Sim}_0\left(1^\lambda, X_{[D]}, N_{[D]}\right), f(X_{[D]}, Y_{[D]})\right).$$
$$\left(\mathsf{View}_1^{\Pi,D}(X_{[D]}, Y_{[D]}), \mathsf{Out}_0^{\Pi,D}(X_{[D]}, Y_{[D]})\right) \stackrel{c}{\approx} \left(\mathsf{Sim}_1\left(1^\lambda, Y_{[D]}, N_{[D]}\right), f(X_{[D]}, Y_{[D]})\right).$$

4 Two-Sided UPSI with Addition

In this section, we present a two-sided UPSI with addition protocol satisfying Definition 3.1 based on the DDH assumption in the random oracle model.

4.1 Construction

Notation. Let $\mathcal{G}$ be a group of prime order q with generator g. Let $H : \{0,1\}^* \to \mathcal{G}$ be a hash function. For a set $X \subseteq \{0,1\}^*$, we denote $H(X) := \{H(x)|x \in X\}$ and $H(X)^k := \{H(x)^k|x \in X\}$.

Construction Overview. Our starting point is the semi-honest PSI protocol based on the DDH assumption [Mea86, HFH99]. The protocol roughly works as follows. Both parties first hash their elements into a group where DDH holds, namely P_0 and P_1 compute $H(X)$ and $H(Y)$ respectively. Each party holds a secret exponentiation key, that is, P_0 holds k_0 and P_1 holds k_1. The parties then use their keys to exponentiate their hashed elements and exchange the results. They further exponentiate the elements in the received set and send back the results. At the end, both parties obtain $H(X)^{k_0k_1}$ and $H(Y)^{k_0k_1}$, from which they can derive the intersection $X \cap Y$.

In the updatable setting, to learn the updated intersection I_d on each Day d, parties only need to learn the update set $I_{\mathsf{update}} = I_d \setminus I_{d-1}$. Observe that I_{update} can be split into two disjoint sets, $I_{X,\mathsf{old}} = X_{\mathsf{old}} \cap Y_d$ (where $X_{\mathsf{old}} := X_{[d-1]} \setminus I_{d-1}$) and $I_{X,\mathsf{new}} = X_d \cap Y_{[d]}$, both of which can be inferred by P_0 from the output of the ideal functionality and its own input. Therefore, it suffices to let P_0 learn both $I_{X,\mathsf{old}}$ and $I_{X,\mathsf{new}}$. Symmetrically, if we let $Y_{\mathsf{old}} := Y_{[d-1]} \setminus I_{d-1}$, then I_{update} can also be split into $I_{Y,\mathsf{old}} = Y_{\mathsf{old}} \cap X_d$ and $I_{Y,\mathsf{new}} = Y_d \cap X_{[d]}$ to allow P_1 to compute the output.

Using the ideas from the above DDH-based protocol, we first ensure that P_0 holds a set $H(X_{\mathsf{old}})^{k_0k_1}$ at the end of Day $(d-1)$, where $X_{\mathsf{old}} = X_{[d-1]} \setminus I_{d-1}$. Then on Day d, P_1 sends $H(Y_d)^{k_1}$ and P_0 computes $H(Y_d)^{k_0k_1}$. From this, P_0 can derive $I_{X,\mathsf{old}} = X_{\mathsf{old}} \cap Y_d$. Symmetrically P_1 can learn $I_{Y,\mathsf{old}} = Y_{\mathsf{old}} \cap X_d$.

The next objective is to let P_0 learn $I_{X,\mathsf{new}} = X_d \cap Y_{[d]}$. Naïvely, the two parties can run a PSI protocol between the two sets X_d and $Y_{[d]}$, but the computational cost of P_1 would grow at least linearly with the size of $Y_{[d]}$, which is unsatisfactory. Observe that $I_{X,\mathsf{new}}$ can also be split into two disjoint sets, $X_d \cap Y_d$ and $X_d \cap Y_{[d-1]}$, the latter being exactly $I_{Y,\mathsf{old}}$. A natural idea is to first run a PSI between X_d and Y_d so that P_0 can learn $X_d \cap Y_d$ and then let P_1 send $I_{Y,\mathsf{old}}$ to P_0. Unfortunately, this idea does not work because it leaks extra information to P_0 (observe that P_0 does not learn $X_d \cap Y_d$ in the ideal world). Nevertheless, we notice that the intersecting elements in $I_{X,\mathsf{new}}$ could only come from either Y_d or $I_{Y,\mathsf{old}}$, both of which are relatively small sets and known to P_1. Therefore, we can let P_0 learn $I_{X,\mathsf{new}}$ by running a PSI with P_1 on the two sets X_d and $Y_d \cup I_{Y,\mathsf{old}}$. In this PSI protocol, P_1 needs to add dummy elements to hide the size of $Y_d \cup I_{Y,\mathsf{old}}$, but the set size is at most $2 \cdot N_d$, hence the PSI is efficient in both computation and communication. The full protocol is described in Figure 4.

4.2 Correctness and Efficiency

Correctness. If both parties follow the protocol honestly, at the end of Day d, we will have the following guarantees with all but negligible probability:

- $I_d = X_{[d]} \cap Y_{[d]}$;
- $X_{\mathsf{old}} = X_{[d]} \setminus I_d$ and $H_X = H(X_{\mathsf{old}})^{k_0k_1}$;
- $Y_{\mathsf{old}} = Y_{[d]} \setminus I_d$ and $H_Y = H(Y_{\mathsf{old}})^{k_0k_1}$.

We prove this by induction. Base case: These guarantees hold on Day 0 since all the sets are initialized as empty sets.

Initialization:
P_0 samples $k_0 \xleftarrow{\$} \mathbb{Z}_q$ and sets $X_{\mathsf{old}} := \emptyset, H_X = \emptyset, I_0 := \emptyset$.
P_1 samples $k_1 \xleftarrow{\$} \mathbb{Z}_q$ and sets $Y_{\mathsf{old}} := \emptyset, H_Y := \emptyset, I_0 := \emptyset$.

Day d**:** Party P_0 inputs a set X_d of size N_d; party P_1 inputs a set Y_d of size N_d.

1. P_0 learns $I_{X,\mathsf{old}} = X_{\mathsf{old}} \cap Y_d$:
   (a) P_1 computes $H(Y_d)^{k_1}$ and sends to P_0.
   (b) On receiving $H(Y_d)^{k_1}$, P_0 raises each element to the power k_0 to obtain $H(Y_d)^{k_0 k_1}$ and compares with H_X (which equals to $H(X_{\mathsf{old}})^{k_0 k_1}$) to learn $I_{X,\mathsf{old}} = X_{\mathsf{old}} \cap Y_d$.
2. Symmetrically, P_1 learns $I_{Y,\mathsf{old}} = Y_{\mathsf{old}} \cap X_d$.
3. Both parties learn the updated intersection:
   (a) P_1 lets $\widetilde{Y_d} := Y_d \cup I_{Y,\mathsf{old}} \cup \widetilde{D_Y}$ where $\widetilde{D_Y}$ consists of dummy random elements so that $|\widetilde{Y_d}| = 2N_d$.
   (b) P_0 and P_1 run a PSI protocol for $\mathcal{F}_{\mathsf{PSI}}$ where P_0's input set is X_d and P_1's input set is $\widetilde{Y_d}$, from which only P_0 learns the output $I_{X,\mathsf{new}}$.
   (c) P_0 computes $I_{\mathsf{update}} := I_{X,\mathsf{new}} \cup I_{X,\mathsf{old}}$ and sends it to P_1.
   (d) Both parties compute $I_d := I_{d-1} \cup I_{\mathsf{update}}$ and output I_d for Day d.
4. P_0 updates X_{old} and H_X:
   (a) P_0 does the following:
   - Let $X'_d := X_d \setminus I_{\mathsf{update}}$ and $\widehat{X'_d} := X'_d \cup \widehat{D_X}$ where $\widehat{D_X}$ consists of dummy random elements so that $|\widehat{X'_d}| = N_d$.
   - Sample a uniform random α from $\mathbb{Z}_q$.
   - Compute $H(\widehat{X'_d})^{\alpha k_0}$ and send to P_1.

   (b) On receiving $H(\widehat{X'_d})^{\alpha k_0}$, P_1 raises each element to the power k_1 to obtain $H(\widehat{X'_d})^{\alpha k_0 k_1}$ and sends back to P_0.
   (c) P_0 does the following:
   - On receiving $H(\widehat{X'_d})^{\alpha k_0 k_1}$, raise each element to the power α^{-1} to obtain $H(\widehat{X'_d})^{k_0 k_1}$, from which derive $H(X'_d)^{k_0 k_1}$.
   - Update $X_{\mathsf{old}} := (X_{\mathsf{old}} \setminus I_{X,\mathsf{old}}) \cup X'_d$ and $H_X := \left(H_X \setminus H(I_{X,\mathsf{old}})^{k_0 k_1}\right) \cup H(X'_d)^{k_0 k_1}$.
5. Symmetrically, P_1 updates $Y_{\mathsf{old}} := (Y_{\mathsf{old}} \setminus I_{Y,\mathsf{old}}) \cup (Y_d \setminus I_{\mathsf{update}})$ and H_Y.

Figure 4: Two-sided UPSI with addition protocol $\Pi_{\mathsf{UPSI\text{-}add\text{-}two}}$.

Induction step: Suppose the guarantees hold on Day $(d-1)$. Let $I_{d-1}, X_{\mathsf{old}}^{(d-1)}, H_X^{(d-1)}, Y_{\mathsf{old}}^{(d-1)}, H_Y^{(d-1)}$ be the sets at the end of Day $(d-1)$. Now we consider Day d with new sets X_d and Y_d. Let $I_d, X_{\mathsf{old}}^{(d)}, H_X^{(d)}, Y_{\mathsf{old}}^{(d)}, H_Y^{(d)}$ be the sets at the end of Day d. In Step 1, party P_0 learns $H(Y_d)^{k_0 k_1}$ and takes the intersection with $H_X^{(d-1)}$ (which equals to $H\left(X_{\mathsf{old}}^{(d-1)}\right)^{k_0 k_1}$). By the collision resistance of the hash function H, the intersection would result in $X_{\mathsf{old}}^{(d-1)} \cap Y_d$ with all but negligible probability, namely $I_{X,\mathsf{old}} = X_{\mathsf{old}}^{(d-1)} \cap Y_d$. Symmetrically, $I_{Y,\mathsf{old}} = Y_{\mathsf{old}}^{(d-1)} \cap X_d$.

In Step 3b, by the correctness of the PSI protocol, the intersection learned by P_0 is

$$\begin{aligned}
I_{X,\mathsf{new}} &= X_d \cap \widetilde{Y_d} = X_d \cap \left(Y_d \cup I_{Y,\mathsf{old}} \cup \widetilde{D_y}\right) \\
&= X_d \cap (Y_d \cup I_{Y,\mathsf{old}}) \qquad \text{(overwhelming probability because } \widetilde{D_y} \text{ are random)} \\
&= (X_d \cap Y_d) \cup (X_d \cap I_{Y,\mathsf{old}}) = (X_d \cap Y_d) \cup \left(X_d \cap \left(Y_{\mathsf{old}}^{(d-1)} \cap X_d\right)\right) \\
&= (X_d \cap Y_d) \cup \left(X_d \cap Y_{\mathsf{old}}^{(d-1)}\right) \\
&= (X_d \cap Y_d) \cup \left(X_d \cap Y_{\mathsf{old}}^{(d-1)}\right) \cup (X_d \cap I_{d-1}) \qquad (X_d \cap I_{d-1} = \emptyset \text{ because } X_d \cap X_{[d-1]} = \emptyset) \\
&= X_d \cap \left(Y_d \cup Y_{\mathsf{old}}^{(d-1)} \cup I_{d-1}\right) = X_d \cap Y_{[d]} \qquad (Y_{\mathsf{old}}^{(d-1)} \cup I_{d-1} = Y_{[d-1]} \text{ by inductive hypothesis})
\end{aligned}$$

The set computed in Step 3c is

$$\begin{aligned}
I_{\mathsf{update}} &= I_{X,\mathsf{new}} \cup I_{X,\mathsf{old}} = \left(X_d \cap Y_{[d]}\right) \cup \left(X_{\mathsf{old}}^{(d-1)} \cap Y_d\right) \\
&= \left(X_d \cap Y_{[d]}\right) \cup \left(X_{\mathsf{old}}^{(d-1)} \cap Y_d\right) \cup (I_{d-1} \cap Y_d) \qquad (I_{d-1} \cap Y_d = \emptyset \text{ because } Y_d \cap Y_{[d-1]} = \emptyset) \\
&= \left(X_d \cap Y_{[d]}\right) \cup \left(\left(X_{\mathsf{old}}^{(d-1)} \cup I_{d-1}\right) \cap Y_d\right) \\
&= \left(X_d \cap Y_{[d]}\right) \cup (X_{[d-1]} \cap Y_d) \qquad (X_{\mathsf{old}}^{(d-1)} \cup I_{d-1} = X_{[d-1]} \text{ by inductive hypothesis})
\end{aligned}$$

Therefore, the new intersection computed in Step 3d is

$$\begin{aligned}
I_d &= I_{d-1} \cup I_{\mathsf{update}} = \left(X_{[d-1]} \cap Y_{[d-1]}\right) \cup \left(X_d \cap Y_{[d]}\right) \cup (X_{[d-1]} \cap Y_d) \\
&= \left(X_{[d-1]} \cap Y_{[d]}\right) \cup \left(X_d \cap Y_{[d]}\right) = X_{[d]} \cap Y_{[d]}.
\end{aligned}$$

In Step 4c, P_0 updates X_{old} as

$$\begin{aligned}
X_{\mathsf{old}}^{(d)} &:= \left(X_{\mathsf{old}}^{(d-1)} \setminus I_{X,\mathsf{old}}\right) \cup (X_d \setminus I_{\mathsf{update}}) \\
&= \left(X_{\mathsf{old}}^{(d-1)} \setminus \left(X_{\mathsf{old}}^{(d-1)} \cap Y_d\right)\right) \cup \left(X_d \setminus \left(\left(X_d \cap Y_{[d]}\right) \cup (X_{[d-1]} \cap Y_d)\right)\right) \\
&= \left(X_{\mathsf{old}}^{(d-1)} \setminus Y_d\right) \cup \left(X_d \setminus Y_{[d]}\right) = \left(\left(X_{[d-1]} \setminus Y_{[d-1]}\right) \setminus Y_d\right) \cup \left(X_d \setminus Y_{[d]}\right) \\
&= \left(X_{[d-1]} \setminus Y_{[d]}\right) \cup \left(X_d \setminus Y_{[d]}\right) = X_{[d]} \setminus Y_{[d]} = X_{[d]} \setminus I_d.
\end{aligned}$$

To update H_X, notice that $I_{X,\mathsf{old}} = X_{\mathsf{old}}^{(d-1)} \cap Y_d \subseteq X_{\mathsf{old}}^{(d-1)}$, thus P_0 can identify $H(I_{X,\mathsf{old}})^{k_0 k_1}$ from $H_X^{(d-1)}$. For $X'_d = X_d \setminus I_{\mathsf{update}}$, P_0 can compute $H(X'_d)^{k_0 k_1}$ in Step 4. Therefore, in Step 4c P_0 obtains $H_X^{(d)} = H\left(X_{\mathsf{old}}^{(d)}\right)^{k_0 k_1}$.

Similarly we can prove these guarantees also hold for $Y_{\mathsf{old}}^{(d)}$ and $H_Y^{(d)}$, which concludes the proof.

Computational and Communication Complexity. On Day d, both parties perform $O(N_d)$ exponentiations and a PSI protocol with set sizes $O(N_d)$. The PSI protocol has both computational and communication complexity $O(N_d)$. Hence the total computational and communication complexity are both $O(N_d)$ and independent of the total set size of each party.

4.3 Security

Theorem 4.1. *Assuming the Decisional Diffie-Hellman (DDH) assumption holds for the group $\mathcal{G}$ and $H(\cdot)$ is modeled as a random oracle, the protocol $\Pi_{\mathsf{UPSI\text{-}add\text{-}two}}$ presented in Figure 4 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{UPSI\text{-}add\text{-}two}}$ (defined in Figure 2) in the $\mathcal{F}_{\mathsf{PSI}}$-hybrid model against semi-honest adversaries.*

Security against corrupted P_0. We construct a PPT Sim_0 that simulates P_0's view as follows. On input $\left(1^\lambda, X_{[D]}, f(X_{[D]}, Y_{[D]})\right)$, where $f(X_{[D]}, Y_{[D]}) := \{I_1, \ldots, I_D\}$ are the outputs of the ideal functionality in the D days, Sim_0 runs the honest P_0 to generate its view and behaves on behalf of an honest P_1 with the following exceptions on each Day $d \in [D]$:

- In Step 1a, let $I'_{X,\mathsf{old}} := X_{[d-1]} \cap (I_d \setminus I_{d-1})$ and compute $H(I'_{X,\mathsf{old}})^{k_1}$. Let R be a set of $N_d - |I'_{X,\mathsf{old}}|$ uniformly randomly sampled group elements in $\mathcal{G}$. Send $H(I'_{X,\mathsf{old}})^{k_1} \cup R$ to P_0 on behalf of P_1.
- In Step 3b, let $I'_{X,\mathsf{new}} := X_d \cap (I_d \setminus I_{d-1})$. Receive P_0's input set as the ideal functionality of $\mathcal{F}_{\mathsf{PSI}}$ and respond to P_0 with $I'_{X,\mathsf{new}}$.
- In Step 5 when P_1 sends $H(\widehat{Y'_d})^{\alpha k_1}$ to P_0 (for a random $\alpha \in \mathbb{Z}_q$), replace it with a set of $|N_d|$ uniformly randomly sampled group elements in $\mathcal{G}$.

Finally Sim_0 outputs P_0's view.

Next we can show that for any $D \in \mathbb{N}^+$, any inputs $(X_{[D]}, Y_{[D]})$,

$$\left(\mathsf{View}_0^{\Pi,D}(X_{[D]}, Y_{[D]}), \mathsf{Out}_1^{\Pi,D}(X_{[D]}, Y_{[D]})\right) \overset{c}{\approx} \left(\mathsf{Sim}_0\left(1^\lambda, X_{[D]}, f(X_{[D]}, Y_{[D]})\right), f(X_{[D]}, Y_{[D]})\right),$$

via a hybrid argument.

Hyb_0 P_0's view and P_1's output in the real protocol.

Hyb_1 Same as Hyb_0 but P_1's output is replaced with $f(X_{[D]}, Y_{[D]})$. This is computationally indistinguishable from Hyb_0 because of the correctness of the protocol shown in Section 4.2.

Hyb_2 Same as Hyb_1 but in Step 3b of each Day $d \in [D]$, let $I'_{X,\mathsf{new}} := X_d \cap (I_d \setminus I_{d-1})$ and let the response from the ideal functionality of $\mathcal{F}_{\mathsf{PSI}}$ to P_0 be $I'_{X,\mathsf{new}}$. We claim that $I_{X,\mathsf{new}} = I'_{X,\mathsf{new}}$.

We show in Section 4.2 that $I_{X,\mathsf{new}} = X_d \cap Y_{[d]}$. Since $X_d \cap X_{[d-1]} = \emptyset$, we have $X_d \cap I_{d-1} = \emptyset$ and hence $(X_d \cap Y_{[d]}) \cap I_{d-1} = \emptyset$. Given that $X_d \cap Y_{[d]} \subseteq I_d$, we have $X_d \cap Y_{[d]} \subseteq I_d \setminus I_{d-1}$, thus $X_d \cap Y_{[d]} \subseteq X_d \cap (I_d \setminus I_{d-1})$, namely $I_{X,\mathsf{new}} \subseteq I'_{X,\mathsf{new}}$. On the other hand, $I_d \subseteq Y_{[d]}$, hence $X_d \cap (I_d \setminus I_{d-1}) \subseteq X_d \cap I_d \subseteq X_d \cap Y_{[d]}$, namely $I'_{X,\mathsf{new}} \subseteq I_{X,\mathsf{new}}$. Therefore $I_{X,\mathsf{new}} = I'_{X,\mathsf{new}}$.

By the correctness of the ideal functionality $\mathcal{F}_{\mathsf{PSI}}$, the two hybrids Hyb_1 and Hyb_2 are computationally indistinguishable.

Hyb_3 Same as Hyb_2 but H is replaced with a random function. This is computationally indistinguishable to Hyb_2 because H is modeled a random oracle.

Hyb_4 Same as Hyb_3 but in Step 1a on each Day $d \in [D]$, for each $y \in Y_d \setminus X_{[d-1]}$, replace $H(y)^{k_1}$ with a uniformly randomly sampled group elements in $\mathcal{G}$. From Hyb_3 to Hyb_4, we actually replace the elements one by one via a sequence of hybrids $\mathsf{Hyb}_{3,0}, \mathsf{Hyb}_{3,1}, \ldots, \mathsf{Hyb}_{3,n}$ where $\mathsf{Hyb}_{3,0} =$

Hyb_3 and $\mathsf{Hyb}_{3,n} = \mathsf{Hyb}_4$. We argue every pair of adjacent hybrids are computationally indistinguishable based on the DDH assumption.

Assume for the purpose of contradiction that there exits a PPT distinguisher $\mathcal{A}$ that can distinguish two adjacent hybrids $\mathsf{Hyb}_{3,i}$ and $\mathsf{Hyb}_{3,i+1}$ where $H(\widetilde{y})^{k_1}$ is replaced by a random group element on some Day d for some $\widetilde{y} \in Y_d \setminus X_{[d-1]}$. We construct a PPT distinguisher $\mathcal{B}$ to break the DDH assumption.

$\mathcal{B}$ is given a tuple of group elements (g_1, g_2, g_3) where $g_1 = g^x, g_2 = g^y$ for random $x, y \in \mathbb{Z}_q$ and g_3 is either g^{xy} or g^z for a random $z \in \mathbb{Z}_q$. $\mathcal{B}$ generates P_0's view as in $\mathsf{Hyb}_{3,i}$ but sets $k_1 := x$ (although x is unknown) and $H(\widetilde{y}) := g_2$.

In particular, whenever $H(\cdot)$ is computed, $\mathcal{B}$ samples a random $r \in \mathbb{Z}_q$ and sets the output to be g^r. In Step 1a when P_1 needs to compute $H(y)^{k_1}$, since $\mathcal{B}$ knows $s \in \mathbb{Z}_q$ such that $H(y) = g^s$, it can compute $H(y)^{k_1}$ as g_1^s; when P_1 samples a random group element for $H(y)^{k_1}$, $\mathcal{B}$ can do the same; for $\widetilde{y}$, $\mathcal{B}$ replaces $H(\widetilde{y})^{k_1}$ with g_3. Since $\widetilde{y} \notin X_{[d-1]}$, we have $\widetilde{y} \notin X_{\mathsf{old}}$ and hence $\widetilde{y} \notin X_{\mathsf{old}} \cap Y_d = I_{X,\mathsf{old}}$ in Step 1b on Day d, thus it doesn't affect P_0's computation.

In Step 4b, to compute $H(x)^{\alpha k_0 k_1}$, since $\mathcal{B}$ knows α, k_0, and $t \in \mathbb{Z}_q$ such that $H(x) = g^t$, it can compute $H(x)^{\alpha k_0 k_1}$ as $g_1^{t\alpha k_0}$. Note that for each x in Step 4b before Day d (not considering the dummy elements), $x \neq \widetilde{y}$ because $\widetilde{y} \notin X_{[d-1]}$; for each x in Step 4b on or after Day d (not considering the dummy elements), $x \neq \widetilde{y}$ because x is not in the intersection. If we take the dummy elements into consideration, then $x \neq \widetilde{y}$ with all but negligible probability, hence $\mathcal{B}$ doesn't have to compute $H(\widetilde{y})^{\alpha k_0 k_1}$.

If $g_3 = g^{xy}$, then $\mathcal{B}$ generates P_0's view as in $\mathsf{Hyb}_{3,i}$; otherwise $\mathcal{B}$ generates P_0's view as in $\mathsf{Hyb}_{3,i+1}$. Since $\mathcal{A}$ can distinguish these two hybrids, $\mathcal{B}$ can break the DDH assumption. Contradiction.

Hyb_5 Same as Hyb_4 but in Step 5 on each Day $d \in [D]$, when P_1 sends $H(\widehat{Y'_d})^{\alpha k_1}$ to P_0 (for a random $\alpha \in \mathbb{Z}_q$), replace it with a set of $|N_d|$ uniformly randomly sampled group elements in $\mathcal{G}$. From Hyb_4 to Hyb_5, we in fact replace the elements one by one via a sequence of hybrids $\mathsf{Hyb}_{4,0}, \mathsf{Hyb}_{4,1}, \ldots, \mathsf{Hyb}_{4,m}$ where $\mathsf{Hyb}_{4,0} = \mathsf{Hyb}_4$ and $\mathsf{Hyb}_{4,m} = \mathsf{Hyb}_5$. We argue that every pair of adjacent hybrids are computationally indistinguishable based on the DDH assumption.

Assume for the purpose of contradiction that there exits a PPT distinguisher $\mathcal{A}$ that can distinguish two adjacent hybrids $\mathsf{Hyb}_{4,i}$ and $\mathsf{Hyb}_{4,i+1}$ where $H(\widehat{y})^{\alpha k_1}$ is replaced with a random group element on some Day d for some $\widehat{y}$. We construct a PPT distinguisher $\mathcal{B}$ to break the DDH assumption.

$\mathcal{B}$ is given a tuple of group elements (g_1, g_2, g_3) where $g_1 = g^x, g_2 = g^y$ for random $x, y \in \mathbb{Z}_q$ and g_3 is either g^{xy} or g^z for a random $z \in \mathbb{Z}_q$. $\mathcal{B}$ generates P_0's view as in $\mathsf{Hyb}_{4,i}$ but in Step 5 sets $\alpha := x$ on behalf of P_1 (although x is unknown) and $H(\widehat{y}) := g_2$.

In particular, whenever $H(\cdot)$ is computed, $\mathcal{B}$ samples a random $r \in \mathbb{Z}_q$ and sets the output as g^r. In Step 5 when P_1 needs to compute $H(y)^{\alpha k_1}$ (where $y \neq \widehat{y}$), since $\mathcal{B}$ knows k_1 as well as $s \in \mathbb{Z}_q$ such that $H(y) = g^s$, it can compute $H(y)^{\alpha k_1}$ as $g_1^{sk_1}$; when P_1 samples a random group element for $H(y)^{\alpha k_1}$ (where $y \neq \widehat{y}$), $\mathcal{B}$ can do the same; for $\widehat{y}$, $\mathcal{B}$ replaces $H(\widehat{y})^{\alpha k_1}$ with $g_3^{k_1}$.

If $g_3 = g^{xy}$, then $\mathcal{B}$ generates P_0's view as in $\mathsf{Hyb}_{3,i}$; otherwise $g_3^{k_1}$ is a random group element, hence $\mathcal{B}$ generates P_0's view as in $\mathsf{Hyb}_{3,i+1}$. Since $\mathcal{A}$ can distinguish these two hybrids, $\mathcal{B}$ can break the DDH assumption. Contradiction.

Hyb_6 Same as Hyb_5 except that H is computed as normal. This is computationally indistinguishable to Hyb_5 because H is modeled as a random oracle.

We claim that P_0's view in this hybrid is exactly Sim_0's output. The only difference between Hyb_6 and Sim_0 is that in Step 1a on each Day $d \in [D]$, $H(y)^{k_1}$ is computed honestly for all $y \in Y_d \cap X_{[d-1]}$ in Hyb_6 while Sim_0 computes $H(y)^{k_1}$ honestly for all $y \in I'_{X,\mathsf{old}}$. We claim that $Y_d \cap X_{[d-1]} = I'_{X,\mathsf{old}}$.

Since $X_{[d-1]} \cap I_d = X_{[d-1]} \cap Y_{[d]}$ and $X_{[d-1]} \cap I_{d-1} = X_{[d-1]} \cap Y_{[d-1]}$, we have $I'_{X,\mathsf{old}} = X_{[d-1]} \cap (I_d \setminus I_{d-1}) = (X_{[d-1]} \cap I_d) \setminus (X_{[d-1]} \cap I_{d-1}) = (X_{[d-1]} \cap Y_{[d]}) \setminus (X_{[d-1]} \cap Y_{[d-1]}) = X_{[d-1]} \cap (Y_{[d]} \setminus Y_{[d-1]}) = X_{[d-1]} \cap Y_d$. This concludes the proof.

Security against corrupted P_1. We construct a PPT Sim_1 that simulates P_1's view as follows. On input $\left(1^\lambda, Y_{[D]}, f(X_{[D]}, Y_{[D]})\right)$, where $f(X_{[D]}, Y_{[D]}) := \{I_1, \ldots, I_D\}$ are the outputs of the ideal functionality in the D days, Sim_1 runs the honest P_1 to generate its view and behaves on behalf of an honest P_0 with the following exceptions on each Day $d \in [D]$:

- In Step 2 when P_0 sends $H(X_d)^{k_0}$ to P_1, let $I'_{Y,\mathsf{old}} := Y_{[d-1]} \cap (I_d \setminus I_{d-1})$ and compute $H(I'_{Y,\mathsf{old}})^{k_0}$. Let R be a set of $N_d - |I'_{Y,\mathsf{old}}|$ uniformly randomly sampled group elements in $\mathcal{G}$. Send $H(I'_{Y,\mathsf{old}})^{k_0} \cup R$ to P_1 on behalf of P_0.
- In Step 3c, let $I'_{\mathsf{update}} := I_d \setminus I_{d-1}$ and send I'_{update} to P_1 on behalf of P_0.
- In Step 4a, send a set of $|N_d|$ uniformly randomly sampled group elements in $\mathcal{G}$ to P_1 on behalf of P_0.

Finally Sim_1 outputs P_1's view.

Next we can show that for any $D \in \mathbb{N}^+$, any inputs $(X_{[D]}, Y_{[D]})$,

$$\left(\mathsf{View}_1^{\Pi,D}(X_{[D]}, Y_{[D]}), \mathsf{Out}_0^{\Pi,D}(X_{[D]}, Y_{[D]})\right) \overset{c}{\approx} \left(\mathsf{Sim}_1\left(1^\lambda, Y_{[D]}, f(X_{[D]}, Y_{[D]})\right), f(X_{[D]}, Y_{[D]})\right),$$

via a hybrid argument.

Hyb_0 P_1's view and P_0's output in the real protocol.

Hyb_1 Same as Hyb_0 but P_0's output is replaced with $f(X_{[D]}, Y_{[D]})$. This is computationally indistinguishable from Hyb_0 because of the correctness of the protocol shown in Section 4.2.

Hyb_2 Same as Hyb_1 but in Step 3c on each Day $d \in [D]$, let $I'_{\mathsf{update}} := I_d \setminus I_{d-1}$ and send I'_{update} to P_1 on behalf of P_0. This is computationally indistinguishable from Hyb_1 because of the correctness of the protocol shown in Section 4.2.

Hyb_3 Same as Hyb_2 but H is replaced with a random function. This is computationally indistinguishable to Hyb_2 because H is modeled a random oracle.

Hyb_4 Same as Hyb_3 but in Step 2 on each Day $d \in [D]$, for each $x \in X_d \setminus Y_{[d-1]}$, replace $H(x)^{k_0}$ with a uniformly randomly sampled group elements in $\mathcal{G}$. This hybrid is computationally indistinguishable from Hyb_3 based on the DDH assumption. The argument is similar to the proof of $\mathsf{Hyb}_3 \overset{c}{\approx} \mathsf{Hyb}_4$ in the security proof against corrupted P_0.

Hyb_5 Same as Hyb_4 but in Step 4a on each Day $d \in [D]$, send a set of $|N_d|$ uniformly randomly sampled group elements in $\mathcal{G}$ to P_1 on behalf of P_0. This hybrid is computationally indistinguishable from Hyb_4 based on the DDH assumption. The argument is similar to the proof of $\mathsf{Hyb}_4 \overset{c}{\approx} \mathsf{Hyb}_5$ in the security proof against corrupted P_0.

Hyb_6 Same as Hyb_5 except that H is computed as normal. This is computationally indistinguishable to Hyb_5 because H is modeled as a random oracle. Finally, we claim that P_1's view in this hybrid is exactly Sim_1's output. The argument is similar to the proof in Hyb_6 of the security proof against corrupted P_0. This concludes the proof.

5 One-Sided UPSI with Addition

In this section, we present a one-sided UPSI with addition protocol satisfying Definition 3.2, where only one party P_0 receives the output on each day.

5.1 Construction

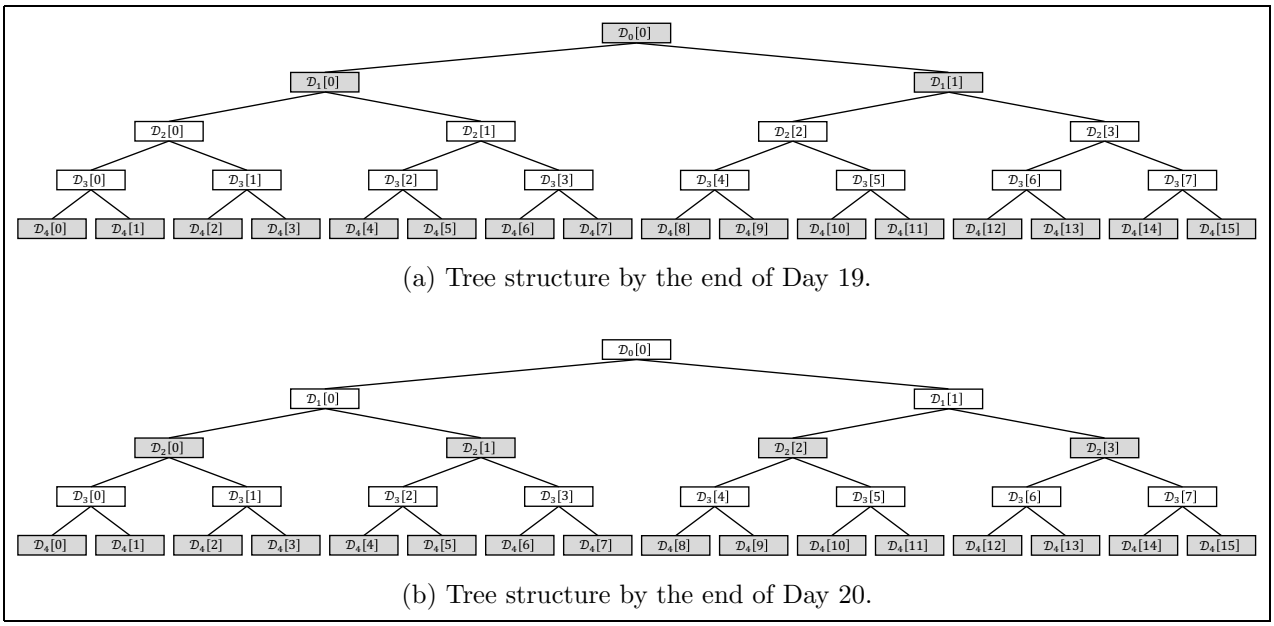
Notation. Let λ be the computational security parameter and σ be the statistical security parameter. Let $\mathcal{G}$ be a group of prime order q with generator g. Let $H_1 : \{0,1\}^* \to \mathcal{G}$ be a hash function and $H_2 : \{0,1\}^* \to \{0,1\}^\lambda$ be another hash function. Let $\mathsf{AHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an additively homomorphic encryption scheme where the message space is a field $\mathbb{F}_p$. For a set $X \subseteq \{0,1\}^*$, we denote $H_1(X) := \{H_1(x) | x \in X\}$ and $H_1(X)^k := \{H_1(x)^k | x \in X\}$. We denote $\mathsf{Enc}_{\mathsf{pk}}(X)$ as $\{\mathsf{Enc}_{\mathsf{pk}}(x) | x \in X\}$.

Let $\mathsf{LS1}(n)$ denote the position of the least significant one in the binary representation of n. In other words, if $n = \sum_{i=0}^{k} b_i \cdot 2^i$, then $\mathsf{LS1}(n) := \min\{i : b_i = 1\}$. For example, $\mathsf{LS1}(7) = 0$ and $\mathsf{LS1}(12) = 2$. For a string $s \in \{0,1\}^\ell$, let $s_{[1..k]}$ (where $1 \le k \le \ell$) be the number whose binary representation is the leading k bits of s. For example, for $s = 010110$, $s_{[1..4]} = (0101)_2 = 5$. In addition, we let $s_{[1..k]} = 0$ for $k = 0$.

Let a node denote a collection of at most 4σ elements (or encrypted elements). For each $i \in \mathbb{N}$, let $\mathcal{D}_i$ denote an array of 2^i nodes on the P_1 side and let $\mathcal{D}_i[j]$ (where $j \in \{0, 1, \ldots, 2^i - 1\}$) be the j-th node in $\mathcal{D}_i$. Similarly, let $\widetilde{\mathcal{D}}_i$ denote an array of 2^i nodes (containing encrypted elements) on the P_0 side and let $\widetilde{\mathcal{D}}_i[j]$ be the j-th node in $\widetilde{\mathcal{D}}_i$.

Construction Overview. For simplicity, we assume $N_d = \sigma$ on each Day d. We discuss how to extend our protocol for $N_d \neq \sigma$ in Section 5.4. Without loss of generality, we assume all the set elements are in the field $\mathbb{F}_p$, namely in the message space of AHE. In case they are not, we can first apply a hash function $H : \{0,1\}^* \to \mathbb{F}_p$ on all the elements.

To learn the updated intersection I_d on each Day d, party P_0 only needs to learn the update set $I_{\mathsf{update}} = I_d \setminus I_{d-1}$. Similar to the previous protocol $\Pi_{\mathsf{UPSI\text{-}add\text{-}two}}$ (see Figure 4), I_{update} can be split into two disjoint sets, $I_{X,\mathsf{old}} = X_{\mathsf{old}} \cap Y_d$ and $I_{X,\mathsf{new}} = X_d \cap Y_{[d]}$ (both can be inferred from the



(a) Tree structure by the end of Day 19.

(b) Tree structure by the end of Day 20.

Figure 5: Example of update on Day 20. A white node indicates it is empty and a gray node indicates it is non-empty. P_1 pushes all the elements in $\mathcal{D}_0$ and $\mathcal{D}_1$, along with the new elements, to $\mathcal{D}_2$.

output in the ideal world). We first use the same approach as in the protocol $\Pi_{\mathsf{UPSI\text{-}add\text{-}two}}$ to let P_0 learn $I_{X,\mathsf{old}}$. Next we describe how to let P_0 learn $I_{X,\mathsf{new}}$ without leaking any information to P_1.

At a high level, P_1 stores all his elements in an encrypted form on P_0's storage in such a way that: (a) P_1 can efficiently and data-obliviously insert new elements to the storage, and (b) P_0 can efficiently query if her element x is in the storage. We construct a binary tree structure to achieve the data obliviousness, efficient data insertion, and efficient data query reminiscent of constructions for oblivous ram (ORAM) [SvDS+18]. In particular, P_1 stores all his elements in a binary tree, which can be updated efficiently when new elements are added to his set. On each day, P_1 updates his tree structure and then sends the corresponding updated encryptions to P_0, which allows her to update the encrypted tree. To query if P_0's element x is in the encrypted tree, P_0 will locate a small set of elements that could possibly contain x. By utilizing additively homomorphic encryption, P_0 is able to learn whether x is among these elements (with P_1's help) without leaking any information about x to P_1.

The binary tree structure works as follows. Initially, the tree is empty. Each node of the tree has a maximum capacity $O(\sigma)$. On each day when there are new elements added to P_1's set, P_1 will insert the new elements into the tree. Intuitively speaking, P_1 starts by adding the new elements to the root of the tree. If the root is full (i.e., reaches the maximum capacity), then P_1 pushes the elements in the root along with the new elements to the second level of the tree. If the second level has any full node, then P_1 pushes all the elements down to the third level. This process continues – if the first L levels of the tree contains any full node, then P_1 pushes all the elements in the first L levels, along with the new elements, to the $(L+1)$-st level of the tree, and then empties the first L levels. For a particular level, an element y is put into a (pseudo-)random node of that level, determined by the output of a hash function $H_2(y)$.

To make the above process *data oblivious* to P_0, P_1 should not wait until exactly when a node is full because that may leak information about P_1's elements. Instead, "pushing" happens in

a *predetermined* way that only depends on P_1's set sizes (which is public to P_0 as well) with the guarantee that no node will reach full capacity except with negligible probability. As an illustration, Figure 5 shows the pushing process on Day $d = 20$, where P_1 pushes all the element in the first two levels of the tree along with the new elements to the third level.

After P_1 updates his local tree structure, he pads every updated node to the maximum capacity using dummy elements and then sends them in an encrypted form to P_0, which allows her to update the encrypted tree structure. Next, when P_0 wishes to query if an element x is in the tree, for each $x \in X_d$, she can first locate a root-to-leaf path of the tree that could possibly contain x (by computing $H_2(x)$). Then, by utilizing additively homomorphic encryption and with the help of P_1, P_0 can learn whether x is contained in any node of the path without learning any more information and without leaking x to P_1. The full protocol is described in Figure 6.

5.2 Correctness

Induction for X_{old}. Observe that, by induction, we can show that at the end of any Day d, $X_{\mathsf{old}} = X_{[d]} \setminus I_d$ and $H_X = H_1(X_{\mathsf{old}})^{k_0 k_1}$. This argument is identical to the one shown in Section 4.2.

Day 1. In Step 1, P_0 learns $\emptyset$ since $X_{\mathsf{old}} = \emptyset$. In Step 2, both parties set $L = \mathsf{max}L = 0$. Then, in Step 3, P_1 inserts the σ elements of Y_1 into $\mathcal{D}_0[0]$. $\mathcal{D}_0[0]$ is then padded to size 4σ before the encrypted node $\widetilde{\mathcal{D}}'_0[0]$ is sent to P_0. In Step 4, P_0 sets $\widetilde{\mathcal{D}}_0[0] = \widetilde{\mathcal{D}}'_0[0]$. In Step 5, for each $x \in X_1$, P_0 computes a pair $(\mathsf{ct}_\alpha, \mathsf{ct}_\beta)$ for each element ct in $\widetilde{\mathcal{D}}_0[0]$. In each pair, $\mathsf{ct}_\beta = \mathsf{Enc}_{\mathsf{pk}_1}(x + \alpha - y)$, where $\mathsf{ct} = \mathsf{Enc}_{\mathsf{pk}_1}(y)$. P_1 decrypts ct_β in each pair and sends back $\mathsf{ct}_r = \mathsf{Enc}_{\mathsf{pk}_0}(\gamma \cdot (x - y))$. Now, P_0 can decrypt this and $\mathsf{Dec}_{\mathsf{sk}_0}(\mathsf{ct}_r) = 0$ if and only if $x \in Y_1$ except with negligible probability (since P_1 also adds random dummy elements). So, P_0 learns $X_1 \cap Y_1$. Finally, in Step 7, P_0 updates X_{old} and H_X similarly as in the previous protocol $\Pi_{\mathsf{UPSI\text{-}add\text{-}two}}$ (Figure 4).

Day d. Now, let's analyze the protocol on any Day d. In Step 1, as a result of the induction-based observation above, P_0 learns $I_{X,\mathsf{old}} = X_{\mathsf{old}} \cap Y_d$ where $X_{\mathsf{old}} = X_{[d-1]} \setminus I_{d-1}$. The data structure $\mathcal{D}$ is a binary tree of depth $\mathsf{max}L$, where each node is of size 4σ. At any i, we denote the 2^i nodes as $\mathcal{D}_i[0], \ldots, \mathcal{D}_i[2^i - 1]$. In Step 3, all levels $i > L$ remain untouched while levels $0, \ldots, L$ are completely revamped. In particular, all the elements (of $Y_{[d-1]}$) at levels up to $(L-1)$ along with the new set Y_d are filled into the nodes at level L. Each element y is inserted into node $\mathcal{D}_L[j]$ where j is the leading L bits of $H_2(y)$. All nodes in levels up to $(L-1)$ are emptied. Finally, as before, these nodes are padded to size 4σ, encrypted and sent to P_0.

Lemma 5.1. *The protocol aborts in Step 3 with negligible probability.*

Proof. The protocol aborts if the size of any node $\mathcal{D}_L[j]$ exceeds 4σ. We show that this happens only with negligible probability. First, assuming hash function $H_2(\cdot)$ is modeled as a random oracle, any element $y \in S$ is equally likely to be placed into any node $\widetilde{\mathcal{D}}_L[j]$ for $j \in \{0, 1, \ldots, 2^L - 1\}$. We now use a couple of sub-lemmas to complete the proof.

Sub-Lemma 5.2. *The number of items inserted into nodes at level L is $|S| = 2^L \cdot \sigma$.*

Proof. Since $L = \mathsf{LS1}(d)$, d is of the form $d = 0 \bmod 2^L$ and $d = 2^L \bmod 2^{L+1}$. Consider $d^* = d - 2^L$, if $d^* > 0$, then $\mathsf{LS1}(d^*) > L$, hence all nodes up to level L are emptied and contain $\emptyset$ on Day d^*. Observe that an element y in P_1's input that is placed on a node in level i is later never placed on

Initialization:

1. P_0 samples $k_0 \xleftarrow{\$} \mathbb{Z}_q$ and sets $X_{\mathsf{old}} := \emptyset, H_X = \emptyset, I_0 := \emptyset$, and $\widetilde{\mathcal{D}}_i[j] := \emptyset$ for all $i \in \mathbb{N}$ and all $j \in \{0, 1, \ldots, 2^i - 1\}$.
   P_1 samples $k_1 \xleftarrow{\$} \mathbb{Z}_q$ and sets $\mathcal{D}_i[j] := \emptyset$ for all $i \in \mathbb{N}$ and all $j \in \{0, 1, \ldots, 2^i - 1\}$.
   Both parties set $\mathsf{max}L := 0$.
2. P_0 generates $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and sends pk_0 to P_1.
   P_1 generates $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and sends pk_1 to P_0.

Day d: P_0 inputs a set X_d of size σ; P_1 inputs a set Y_d of size σ.

1. P_0 learns $I_{X,\mathsf{old}} = X_{\mathsf{old}} \cap Y_d$:
   (a) P_1 computes $H_1(Y_d)^{k_1}$ and sends to P_0.
   (b) On receiving $H_1(Y_d)^{k_1}$, P_0 raises each element to the power k_0 to obtain $H_1(Y_d)^{k_0 k_1}$ and compares with H_X (which equals to $H_1(X_{\mathsf{old}})^{k_0 k_1}$) to learn $I_{X,\mathsf{old}} = X_{\mathsf{old}} \cap Y_d$.
2. Both parties let $L := \mathsf{LS1}(d)$ and $\mathsf{max}L := \max\{L, \mathsf{max}L\}$.
3. P_1 updates $\mathcal{D}$ by doing the following:
   (a) Let $S := \left(\bigcup_{i=0}^{L-1} \bigcup_{j=0}^{2^i-1} \mathcal{D}_i[j]\right) \cup Y_d$.
   (b) For each $i \in \{0, 1, \ldots, L\}$ and for each $j \in \{0, 1, \ldots, 2^i - 1\}$, set $\mathcal{D}_i[j] := \emptyset$.
   (c) For each element $y \in S$, let $j := H_2(y)_{[1..L]}$ and add y into the node $\mathcal{D}_L[j]$. If the size of $\mathcal{D}_L[j]$ exceeds 4σ, then abort.
   (d) For each $j \in \{0, 1, \ldots, 2^L - 1\}$, construct a node $\mathcal{D}'_L[j]$ of size 4σ by padding $\mathcal{D}_L[j]$ with dummy random elements. Compute $\widetilde{\mathcal{D}}'_L[j] \leftarrow \mathsf{Enc}_{\mathsf{pk}_1}(\mathcal{D}'_L[j])$.
   (e) Send $\left\{\widetilde{\mathcal{D}}'_L[j]\right\}_{j \in \{0,1,\ldots,2^L-1\}}$ to P_0.
4. P_0 updates $\widetilde{\mathcal{D}}$ by doing the following:
   (a) For each $i \in \{0, 1, \ldots, L-1\}$ and for each $j \in \{0, 1, \ldots, 2^i - 1\}$, set $\widetilde{\mathcal{D}}_i[j] := \emptyset$.
   (b) For each $j \in \{0, 1, \ldots, 2^L - 1\}$, set $\widetilde{\mathcal{D}}_L[j] := \widetilde{\mathcal{D}}'_L[j]$.
5. P_0 learns $I_{X,\mathsf{new}} = X_d \cap Y_{[d]}$:
   P_0 first sets $I_{X,\mathsf{new}} := \emptyset$. Then for each $x \in X_d$:
   (a) P_0 does the following:
      i. Set $\mathcal{C}_0 := \emptyset$.
      ii. For each $i \in \{0, \ldots, \mathsf{max}L\}$, let $j := H_2(x)_{[1..i]}$; if $\widetilde{\mathcal{D}}_i[j] \neq \emptyset$, then for each $\mathsf{ct} \in \widetilde{\mathcal{D}}_i[j]$:
      Sample $\alpha \xleftarrow{\$} \mathbb{F}_p$, compute $\mathsf{ct}_\alpha \leftarrow \mathsf{Enc}_{\mathsf{pk}_0}(\alpha)$ and $\mathsf{ct}_\beta \leftarrow \mathsf{Enc}_{\mathsf{pk}_1}(x + \alpha) \ominus \mathsf{ct}$, and add a pair $(\mathsf{ct}_\alpha, \mathsf{ct}_\beta)$ to $\mathcal{C}_0$.
      iii. Send $\mathcal{C}_0$ to P_1.
   (b) P_1 does the following:
      i. Set $\mathcal{C}_1 := \emptyset$.
      ii. For each pair $(\mathsf{ct}_\alpha, \mathsf{ct}_\beta) \in \mathcal{C}_0$, sample $\gamma \xleftarrow{\$} \mathbb{F}_p$, compute $\beta \leftarrow \mathsf{Dec}_{\mathsf{sk}_1}(\mathsf{ct}_\beta)$, $\mathsf{ct}_r \leftarrow \gamma \odot (\mathsf{Enc}_{\mathsf{pk}_0}(\beta) \ominus \mathsf{ct}_\alpha)$ and add ct_r to $\mathcal{C}_1$.
      iii. Send $\mathcal{C}_1$ in a randomly permuted order to P_0.
   (c) P_0 does the following:
   For each $\mathsf{ct}_r \in \mathcal{C}_1$, compute $r \leftarrow \mathsf{Dec}_{\mathsf{sk}_0}(\mathsf{ct}_r)$. Add x to the set $I_{X,\mathsf{new}}$ if $r = 0$.
6. P_0 computes $I_d := I_{d-1} \cup I_{X,\mathsf{old}} \cup I_{X,\mathsf{new}}$ and output I_d for Day d.
7. P_0 updates X_{old} and H_X:
   (a) P_0 does the following:
   Let $X'_d := X_d \setminus I_d$ and construct $\widehat{X'_d}$ of size σ by padding X_d with dummy random elements.
   Sample $\alpha \xleftarrow{\$} \mathbb{Z}_q$, compute $H_1(\widehat{X'_d})^{\alpha k_0}$ and send to P_1.
   (b) P_1 raises each element in $H_1(\widehat{X'_d})^{\alpha k_0}$ to the power k_1 to obtain $H_1(\widehat{X'_d})^{\alpha k_0 k_1}$ and sends back to P_0.
   (c) P_0 raises each element in $H_1(\widehat{X'_d})^{\alpha k_0 k_1}$ to the power α^{-1} to obtain $H_1(\widehat{X'_d})^{k_0 k_1}$, from which it derives $H_1(X'_d)^{k_0 k_1}$. Then P_0 updates $X_{\mathsf{old}} := (X_{\mathsf{old}} \setminus I_{X,\mathsf{old}}) \cup X'_d$ and $H_X := \left(H_X \setminus H_1(I_{X,\mathsf{old}})^{k_0 k_1}\right) \cup H_1(X'_d)^{k_0 k_1}$.

Figure 6: One-sided UPSI with addition protocol $\Pi_{\mathsf{UPSI\text{-}add\text{-}one}}$.

level i_1 where $i_1 < i$. In particular, elements either remain at the same level or are pushed further down the tree on each day. From Day $(d^* + 1, \ldots, d-1)$, the number of elements added to the tree is $\sigma \cdot (d - 1 - d^*) = \sigma \cdot (2^L - 1)$. The number of elements added on Day d is σ and hence $|S| = 2^L \cdot \sigma$. □

Sub-Lemma 5.3. *Given $N = \mathsf{Poly}(\sigma)$ balls distributed into $\frac{N}{\sigma}$ bins, where every ball is equally likely to be placed in any bin, $\Pr[\text{size of every bin} \leq 4\sigma] \geq 1 - \mathsf{negl}(\sigma)$.*

Proof. Let $X_{i,\geq k}$ be an indicator variable that the i-th node has at least k balls. Then

$$\begin{aligned}\Pr[X_{i,\geq k} = 1] &\leq \binom{N}{k} \frac{1}{(N/\sigma)^k} \leq \left(\frac{N}{N/\sigma}\right)^k \frac{1}{k!} = \sigma^k \frac{1}{k!} \\ &\leq \sigma^k \frac{1}{\sqrt{2\pi k}\left(\frac{k}{e}\right)^k} \quad \text{(using Stirling's approximation)}\end{aligned}$$

Let $k = 4\sigma$, then we have $\Pr[X_{i,\geq k} = 1] \leq \frac{\sigma^k}{\left(\frac{4\sigma}{e}\right)^k} = \frac{1}{(4/e)^{4\sigma}} \leq \frac{1}{2^{2\sigma}}$. By taking a union bound we have

$$\Pr\left[\exists i \in N, X_{i,\geq 4\sigma} = 1\right] \leq \frac{N}{2^{2\sigma}} = \mathsf{negl}(\sigma).$$

□

Combining the above two sub-lemmas, it is easy to see that no node $\mathcal{D}_L[j]$ has size more than 4σ except with negligible probability. □

In Step 4, P_0 updates the encrypted database $\widetilde{\mathcal{D}}$. In Step 5, for each $x \in X_d$, for each $i \in \{0, 1, \ldots, \mathsf{max}L\}$, P_0 computes $H_2(x)_{[1,\ldots,i]}$ to identify which nodes of $\widetilde{\mathcal{D}}$ (at each level) could possibly contain x. Then, for each such non-empty node, for each ciphertext ct in it, ($\mathsf{ct} = \mathsf{Enc}_{\mathsf{pk}_1}(y)$ where $y \in Y_{[d]}$ or y is a random dummy element), P_0 computes and sends $(\mathsf{Enc}_{\mathsf{pk}_0}(\alpha), \mathsf{Enc}_{\mathsf{pk}_1}(x - y - \alpha))$. P_1 responds back with $\mathsf{Enc}_{\mathsf{pk}_0}(\gamma(x-y))$ which P_0 can decrypt. Observe that this is 0 if and only if $x = y$ where $y \in Y_{[d]}$ except with negligible probability (if y equals a random dummy element). In this manner, P_0 learns whether each element $x \in X_d$ belongs to $Y_{[d]}$ and computes $I_{X,\mathsf{new}} = X_d \cap Y_{[d]}$. Finally, in Step 7, P_0 updates X_{old} and H_X as done in the previous protocol $\Pi_{\mathsf{UPSI\text{-}add\text{-}two}}$ (Figure 4).

5.3 Security

Theorem 5.4. *Given an additively homomorphic encryption scheme AHE, assuming that the Decisional Diffie-Hellman (DDH) assumption holds for the group $\mathcal{G}$, and that H_1, H_2 are modeled as random oracles, the protocol $\Pi_{\mathsf{UPSI\text{-}add\text{-}one}}$ presented in Figure 6 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{UPSI\text{-}add\text{-}one}}$ (defined in Figure 2) against semi-honest adversaries.*

Security against corrupted P_0. First, let $\mathsf{Num1}(n)$ denote the number of 1's in the binary representation of n. In other words, if $n = \sum_{i=0}^{k} b_i \cdot 2^i$, then $\mathsf{Num1}(n) := |\{i : b_i = 1\}|$. For example, $\mathsf{Num1}(7) = 3$ and $\mathsf{Num1}(12) = 2$. We construct a PPT Sim_0 that simulates P_0's view as follows. On input $\left(1^\lambda, X_{[D]}, f(X_{[D]}, Y_{[D]})\right)$, where $f(X_{[D]}, Y_{[D]}) := \{I_1, \ldots, I_D\}$ are the outputs of the ideal functionality in the D days, Sim_0 runs the honest P_0 to generate its view and behaves on behalf of an honest P_1 with the following exceptions on each Day $d \in [D]$:

- In Step 1a, let $I'_{X,\mathsf{old}} := X_{[d-1]} \cap (I_d \setminus I_{d-1})$ and compute $H_1(I'_{X,\mathsf{old}})^{k_1}$. Let R be a set of $(\sigma - |I'_{X,\mathsf{old}}|)$ randomly sampled group elements in $\mathcal{G}$. Send $H_1(I'_{X,\mathsf{old}})^{k_1} \cup R$ to P_0 on behalf of P_1.
- In Step 3c, never abort on behalf of P_1.
- In Step 3e, let $\widetilde{\mathcal{D}}'_L[j]$ be a set of 4σ encryptions of 0 under pk_1, namely $\mathsf{Enc}_{\mathsf{pk}_1}(0)$. Send $\left\{\widetilde{\mathcal{D}}'_L[j]\right\}_{j\in\{0,1,\ldots,2^L-1\}}$ to P_0 on behalf of P_1.
- In Step 5(b)iii, if $x \notin I_d$, let $\mathcal{C}_1$ be a set of $4\sigma \cdot \mathsf{Num1}(d)$ encryptions of random elements under pk_0, namely $\mathsf{Enc}_{\mathsf{pk}_0}(r)$ for $r \xleftarrow{\$} \mathbb{F}_p$; otherwise, let $\mathcal{C}_1$ be a set containing $\mathsf{Enc}_{\mathsf{pk}_0}(0)$ and $(4\sigma\cdot\mathsf{Num1}(d)-1)$ encryptions of random elements under pk_0. Send $\mathcal{C}_1$ in a randomly permuted order to P_0 on behalf of P_1.

Finally, Sim_0 outputs P_0's view.

We show that for any $D \in \mathbb{N}^+$, any inputs $(X_{[D]}, Y_{[D]})$,

$$\mathsf{View}_0^{\Pi,D}(X_{[D]}, Y_{[D]}) \overset{c}{\approx} \mathsf{Sim}_0\left(1^\lambda, X_{[D]}, f(X_{[D]}, Y_{[D]})\right)$$

via a hybrid argument.

Hyb_0 P_0's view in the real protocol.

Hyb_1 Same as Hyb_1 but H_1 is replaced with a random function. This is computationally indistinguishable to Hyb_0 because H_1 is modeled a random oracle.

Hyb_2 Same as Hyb_1 but in Step 1a on each Day $d \in [D]$, for each $y \in Y_d \setminus X_{[d-1]}$, replace $H_1(y)^{k_1}$ with a uniformly randomly sampled group elements in $\mathcal{G}$. From Hyb_1 to Hyb_2, we actually replace the elements one by one via a sequence of hybrids $\mathsf{Hyb}_{1,0}, \mathsf{Hyb}_{1,1}, \ldots, \mathsf{Hyb}_{1,n}$ where $\mathsf{Hyb}_{1,0} = \mathsf{Hyb}_1$ and $\mathsf{Hyb}_{1,n} = \mathsf{Hyb}_2$. We argue every pair of adjacent hybrids are computationally indistinguishable based on the DDH assumption.

Assume for the purpose of contradiction that there exits a PPT distinguisher $\mathcal{A}$ that can distinguish two adjacent hybrids $\mathsf{Hyb}_{1,i}$ and $\mathsf{Hyb}_{1,i+1}$ where $H_1(\widetilde{y})^{k_1}$ is replaced by a random group element on some Day d for some $\widetilde{y} \in Y_d \setminus X_{[d-1]}$. We construct a PPT distinguisher $\mathcal{B}$ to break the DDH assumption.

$\mathcal{B}$ is given a tuple of group elements (g_1, g_2, g_3) where $g_1 = g^x, g_2 = g^y$ for random $x, y \in \mathbb{Z}_q$ and g_3 is either g^{xy} or g^z for a random $z \in \mathbb{Z}_q$. $\mathcal{B}$ generates P_0's view as in $\mathsf{Hyb}_{1,i}$ but sets $k_1 := x$ (although x is unknown) and $H_1(\widetilde{y}) := g_2$.

In particular, whenever $H_1(\cdot)$ is computed, $\mathcal{B}$ samples a random $r \in \mathbb{Z}_q$ and sets the output to be g^r. In Step 1a when P_1 needs to compute $H_1(y)^{k_1}$, since $\mathcal{B}$ knows $s \in \mathbb{Z}_q$ such that $H_1(y) = g^s$, it can compute $H_1(y)^{k_1}$ as g_1^s; when P_1 samples a random group element for $H_1(y)^{k_1}$, $\mathcal{B}$ can do the same; for $\widetilde{y}$, $\mathcal{B}$ replaces $H_1(\widetilde{y})^{k_1}$ with g_3. Since $\widetilde{y} \notin X_{[d-1]}$, we have $\widetilde{y} \notin X_{\mathsf{old}}$ and hence $\widetilde{y} \notin X_{\mathsf{old}} \cap Y_d = I_{X,\mathsf{old}}$ in Step 1b on Day d, thus it doesn't affect P_0's computation.

In Step 7b, to compute $H_1(x)^{\alpha k_0 k_1}$, since $\mathcal{B}$ knows α, k_0, and $t \in \mathbb{Z}_q$ such that $H_1(x) = g^t$, it can compute $H_1(x)^{\alpha k_0 k_1}$ as $g_1^{t\alpha k_0}$. Note that for each x in Step 7b before Day d (not considering the dummy elements), $x \neq \widetilde{y}$ because $\widetilde{y} \notin X_{[d-1]}$; for each x in Step 7b on or after

Day d (not considering the dummy elements), $x \neq \widetilde{y}$ because x is not in the intersection. If we take the dummy elements into consideration, then $x \neq \widetilde{y}$ with all but negligible probability, hence $\mathcal{B}$ doesn't have to compute $H_1(\widetilde{y})^{\alpha k_0 k_1}$.

If $g_3 = g^{xy}$, then $\mathcal{B}$ generates P_0's view as in $\mathsf{Hyb}_{1,i}$; otherwise $\mathcal{B}$ generates P_0's view as in $\mathsf{Hyb}_{1,i+1}$. Since $\mathcal{A}$ can distinguish these two hybrids, $\mathcal{B}$ can break the DDH assumption. Contradiction.

Hyb_3 Same as Hyb_2 but in Step 3c, P_1 never aborts. By Lemma 5.1, the probability that P_1 aborts is negligible, hence this hybrid is computationally indistinguishable from Hyb_2.

Hyb_4 Same as Hyb_3 except that in Step 5(b)iii on each day $d \in [D]$, replace each ct_r by a fresh encryption of $\gamma \cdot (\beta - \alpha)$ under pk_0. This hybrid is statistically indistinguishable from Hyb_3 by the re-randomization property of the additively homomorphic encryption scheme. In particular, the encryption ct_r computed from $(\mathsf{ct}_\alpha, \beta, \gamma)$ by homomorphic operations is statistically indistinguishable from a fresh encryption of r even given the secret key sk_0.

Hyb_5 Same as Hyb_4 except that in Step 5(b)iii on each day $d \in [D]$, if $x \notin I_d$, let $\mathcal{C}_1$ be a set of $4\sigma \cdot \mathsf{Num1}(d)$ encryptions of random elements under pk_0, namely $\mathsf{Enc}_{\mathsf{pk}_0}(r)$ for $r \xleftarrow{\$} \mathbb{F}_p$. Send $\mathcal{C}_1$ in a randomly permuted order to P_0 on behalf of P_1.

First, our construction guarantees that on each day $d \in [D]$, there are exactly $\mathsf{Num1}(d)$ levels of the tree that are non-empty, hence the size of $\mathcal{C}_0$ is $4\sigma \cdot \mathsf{Num1}(d)$. If $x \notin I_d$, then for each pair $(\mathsf{ct}_\alpha, \mathsf{ct}_\beta) \in \mathcal{C}_0$, we know that $\beta - \alpha = x - y = 0$ with negligible probability (note that some y values are randomly sampled by P_1, so the probability is not 0 but negligible). In case $\beta - \alpha \neq 0$, then $\gamma \cdot (\beta - \alpha)$ for a random $\gamma \xleftarrow{\$} \mathbb{F}_p$ is identically distributed from a random value $r \xleftarrow{\$} \mathbb{F}_p$.

Therefore, this hybrid is statistically indistinguishable from Hyb_4.

Hyb_6 Same as Hyb_5 except that in Step 5(b)iii on each day $d \in [D]$, if $x \in I_d$, let $\mathcal{C}_1$ be a set containing $\mathsf{Enc}_{\mathsf{pk}_0}(0)$ and $(4\sigma \cdot \mathsf{Num1}(d) - 1)$ encryptions of random elements under pk_0. Send $\mathcal{C}_1$ in a randomly permuted order to P_0 on behalf of P_1.

If $x \in I_d$, then there exists one pair $(\mathsf{ct}_\alpha, \mathsf{ct}_\beta) \in \mathcal{C}_0$ such that $\beta - \alpha = 0$; for all other pairs, $\beta - \alpha = 0$ with negligible probability. For the pair such that $\beta - \alpha = 0$, $\gamma \cdot (\beta - \alpha) = 0$ for any γ. For all other pairs, in case $\beta - \alpha \neq 0$, then $\gamma \cdot (\beta - \alpha)$ for a random $\gamma \xleftarrow{\$} \mathbb{F}_p$ is identically distributed from a random value $r \xleftarrow{\$} \mathbb{F}_p$.

Therefore, this hybrid is statistically indistinguishable from Hyb_5.

Hyb_7 Same as Hyb_6 but in Step 3e on each day $d \in [D]$, let $\widetilde{\mathcal{D}}'_L[j]$ be a set of 4σ encryptions of 0 under pk_1, namely $\mathsf{Enc}_{\mathsf{pk}_1}(0)$. Send $\left\{\widetilde{\mathcal{D}}'_L[j]\right\}_{j \in \{0,1,\ldots,2^L-1\}}$ to P_0 on behalf of P_1. This hybrid is computationally indistinguishable from Hyb_6 by the CPA security of the additively homomorphic encryption scheme.

Hyb_8 Same as Hyb_7 except that H_1 is computed as normal. This is computationally indistinguishable to Hyb_7 because H_1 is modeled as random oracles.

We claim that P_0's view in this hybrid is exactly Sim_0's output. The only difference between Hyb_8 and Sim_0 is that in Step 1a on each Day $d \in [D]$, $H_1(y)^{k_1}$ is computed honestly for all $y \in Y_d \cap X_{[d-1]}$ in Hyb_8 while Sim_0 computes $H_1(y)^{k_1}$ honestly for all $y \in I'_{X,\mathsf{old}}$. We claim that $Y_d \cap X_{[d-1]} = I'_{X,\mathsf{old}}$.

Since $X_{[d-1]} \cap I_d = X_{[d-1]} \cap Y_{[d]}$ and $X_{[d-1]} \cap I_{d-1} = X_{[d-1]} \cap Y_{[d-1]}$, we have $I'_{X,\mathsf{old}} = X_{[d-1]} \cap (I_d \setminus I_{d-1}) = (X_{[d-1]} \cap I_d) \setminus (X_{[d-1]} \cap I_{d-1}) = (X_{[d-1]} \cap Y_{[d]}) \setminus (X_{[d-1]} \cap Y_{[d-1]}) = X_{[d-1]} \cap (Y_{[d]} \setminus Y_{[d-1]}) = X_{[d-1]} \cap Y_d$. This concludes the proof.

Security against corrupted P_1. We construct a PPT Sim_1 that simulates P_1's view as follows. On input $\left(1^\lambda, Y_{[D]}\right)$, Sim_1 runs the honest P_1 to generate its view and behaves on behalf of an honest P_0 with the following exceptions on each Day $d \in [D]$:

- In Step 5(a)iii, let $\mathcal{C}_0$ be a set of $4\sigma \cdot \mathsf{Num1}(d)$ pairs of encryptions $(\mathsf{ct}_\alpha, \mathsf{ct}_\beta)$, where $\mathsf{ct}_\alpha \leftarrow \mathsf{Enc}_{\mathsf{pk}_0}(0)$ and $\mathsf{ct}_\beta \leftarrow \mathsf{Enc}_{\mathsf{pk}_1}(r)$ for $r \xleftarrow{\$} \mathbb{F}_p$. Send $\mathcal{C}_0$ to P_1 on behalf of P_0.
- In Step 7a, send a set of σ randomly sampled group elements in $\mathcal{G}$ to P_1 on behalf of P_0.

Finally Sim_1 outputs P_1's view.

Next we show that for any $D \in \mathbb{N}^+$, any inputs $(X_{[D]}, Y_{[D]})$,

$$\left(\mathsf{View}_1^{\Pi,D}(X_{[D]}, Y_{[D]}), \mathsf{Out}_0^{\Pi,D}(X_{[D]}, Y_{[D]})\right) \overset{c}{\approx} \left(\mathsf{Sim}_1\left(1^\lambda, Y_{[D]}\right), f(X_{[D]}, Y_{[D]})\right).$$

via a hybrid argument.

Hyb_0 P_1's view and P_0's output in the real protocol.

Hyb_1 Same as Hyb_0 but P_0's output is replaced with $f(X_{[D]}, Y_{[D]})$. This is computationally indistinguishable from Hyb_0 because of the correctness of the protocol shown in Section 5.2.

Hyb_2 Same as Hyb_1 but H_1 is replaced with a random function. This is computationally indistinguishable to Hyb_1 because H_1 is modeled a random oracle.

Hyb_3 Same as Hyb_2 but in Step 7a on each Day $d \in [D]$, send a set of σ randomly sampled group elements in $\mathcal{G}$ to P_1 on behalf of P_0. This hybrid is computationally indistinguishable from Hyb_2 based on the DDH assumption. The argument is similar to the proof of $\mathsf{Hyb}_1 \overset{c}{\approx} \mathsf{Hyb}_2$ in the security proof against corrupted P_0.

Hyb_4 Same as Hyb_3 except that in Step 5(a)iii on each day $d \in [D]$, replace each ct_β by a fresh encryption of $(x + \alpha - y)$ under pk_1. This hybrid is statistically indistinguishable from Hyb_3 by the re-randomization property of the additively homomorphic encryption scheme.

Hyb_5 Same as Hyb_4 except that in Step 5(a)iii on each day $d \in [D]$, replace each ct_α by a fresh encryption of 0 under pk_0. This hybrid is computationally indistinguishable from Hyb_4 by the CPA security of the additively homomorphic encryption scheme.

Hyb_6 Same as Hyb_5 except that in Step 5(a)iii on each day $d \in [D]$, replace each ct_β by a fresh encryption of r for a random $r \xleftarrow{\$} \mathbb{F}_p$ under pk_1. We know that $(x + \alpha - y)$ is identically distributed from a random value $r \xleftarrow{\$} \mathbb{F}_p$. Hence this hybrid is identically indistinguishable from Hyb_5.

Our construction guarantees that on each day $d \in [D]$, there are exactly $\mathsf{Num1}(d)$ levels of the tree that are non-empty, hence the size of $\mathcal{C}_0$ is $4\sigma \cdot \mathsf{Num1}(d)$. Thus, in this hybrid, $\mathcal{C}_0$ contains $4\sigma \cdot \mathsf{Num1}(d)$ pairs of encryptions $(\mathsf{ct}_\alpha, \mathsf{ct}_\beta)$, where $\mathsf{ct}_\alpha \leftarrow \mathsf{Enc}_{\mathsf{pk}_0}(0)$ and $\mathsf{ct}_\beta \leftarrow \mathsf{Enc}_{\mathsf{pk}_1}(r)$ for $r \xleftarrow{\$} \mathbb{F}_p$.

Hyb_7 Same as Hyb_6 except that H_1 is computed as normal. This is computationally indistinguishable to Hyb_6 because H_1 is modeled as a random oracle. P_1's view in this hybrid is exactly Sim_1's output. This concludes the proof.

5.4 Extension

In this section, we extend our protocol to the general setting when the number of elements added by both parties on any day $N_d \neq \sigma$. For simplicity, let's assume N_d is a multiple of σ (we can always pad with dummy elements to make it a multiple of σ).[2] The high level idea is to split the input into N_d/σ batches of length σ and essentially run the basic protocol over multiple days, with σ elements as input on each day. We use a separate counter d^* to track the "day number" of the basic protocol. We provide more details below.

Let $X_d = \{X'_{d^*}, X'_{d^*+1}, \ldots, X'_{d^*+N_d/\sigma-1}\}$, $Y_d = \{Y'_{d^*}, Y'_{d^*+1}, \ldots, Y'_{d^*+N_d/\sigma-1}\}$ be the two input sets split into N_d/σ batches of length σ each. First, run the basic protocol on a fresh day (day d^* for the basic protocol) with inputs X'_{d^*} and Y'_{d^*} respectively. Let's call this sub-day d^* to indicate that this is the counter for the underlying basic protocol. Then, run the basic protocol on sub-day (d^*+1) with inputs X'_{d^*+1} and Y'_{d^*+1}. Repeat this till the basic protocol is run on sub-day $(d^*+N_d/\sigma-1)$ using inputs $X'_{d^*+N_d/\sigma-1}$ and $Y'_{d^*+N_d/\sigma-1}$. Finally, before moving to day $(d+1)$ where both parties have fresh inputs, we update $d^* = (d^*+N_d/\sigma)$. (If $N_d = \sigma$, we would have $d^* = d^*+1$ as in the basic protocol). While this is the high level approach, unfortunately, the protocol does *not* quite work. Briefly, running the basic protocol on N_d/σ sub-days leaks the intermediate output each time and this is undesirable (and not leaked in the ideal world). Instead, the idea is to run the steps where the "actual intersection" is computed only once across these many batches.

In more detail, Step 1 is not run on each sub-day from d^* to $(d^*+N_d/\sigma-1)$ – instead, we run the step only on day d^* with P_1's input as Y_d (and not only Y'_{d^*}) to allow P_0 to learn $X_{\mathsf{old}} \cap Y_d$. Observe that Step 1 does not require either party's set to be of size σ. Next, Step 5 is also not run on each sub-day – we run steps Step 3 and Step 4 on each sub-day to update the database and finally, only on day $(d^*+N_d/\sigma-1)$, execute Step 5 (with P_0 using entire input set X_d) to allow P_0 to compute $X_d \cap Y_{[d]}$. Further, as an optimization, even in Step 3, P_1 need not send the encrypted database $\left\{\widetilde{\mathcal{D}}'_L[j]\right\}_{j\in\{0,1,\ldots,2^L-1\}}$ on each sub-day. Consider two sub-days d_a, d_b with $d_a < d_b$ and $L = \mathsf{LS1}(d_a) = \mathsf{LS1}(d_b)$. Now, on sub-day d_b, all the updates to $\left\{\widetilde{\mathcal{D}}'_L[j]\right\}$ overwrite the updates $\left\{\widetilde{\mathcal{D}}'_L[j]\right\}$ made to the database on sub-day d_a. In particular, the elements written to the database on sub-day d_a are pushed a level down the tree before the updates on sub-day d_b are recorded on level L. Building on this idea, let $\mathsf{max}L_d$ be the maximum value of L over the sub-days $d^*, d^*+1, \ldots, (d^*+N_d/\sigma-1)$. At the end of sub-day $(d^*+N_d/\sigma-1)$, P_0 sends $\left\{\widetilde{\mathcal{D}}'_L[j]\right\}_{j\in\{0,1,\ldots,2^L-1\}}$

[2] We note that P_0's input size actually need not be padded to a multiple of σ because she can make queries for each element $x \in X_d$ independently.

Initialization: Same as Figure 6. Also, both parties set $d^* := 1$.

Day d**:** P_0 inputs a set X_d of size N_d; P_1 inputs a set Y_d of size N_d.
Let $X_d = \{X'_{d^*}, X'_{d^*+1}, \ldots, X'_{d^*+N_d/\sigma-1}\}$ and $Y_d = \{Y'_{d^*}, Y'_{d^*+1}, \ldots, Y'_{d^*+N_d/\sigma-1}\}$ where each X'_i, Y'_i is of size σ.

1. P_0 learns $I_{X,\mathsf{old}} = X_{\mathsf{old}} \cap Y_d$:
   (a) P_1 computes $H_1(Y_d)^{k_1}$ and sends to P_0.
   (b) On receiving $H_1(Y_d)^{k_1}$, P_0 raises each element to the power k_0 to obtain $H_1(Y_d)^{k_0 k_1}$ and compares with H_X (which equals to $H_1(X_{\mathsf{old}})^{k_0 k_1}$) to learn $I_{X,\mathsf{old}} = X_{\mathsf{old}} \cap Y_d$.
2. Both parties set $\mathsf{max}L_d = 0$.
3. For each $t \in \{d^*, d^*+1, \ldots, (d^* + N_d/\sigma - 1)\}$, do the following:
   (a) Both parties set $L := \mathsf{LS1}(t)$, $\mathsf{max}L := \max\{L, \mathsf{max}L\}$, $\mathsf{max}L_d = \max(L, \mathsf{max}L_d)$.
   (b) P_1 updates $\mathcal{D}$ by doing the following:
      i. Let $S := \left(\bigcup_{i=0}^{L-1} \bigcup_{j=0}^{2^i-1} \mathcal{D}_i[j]\right) \cup Y'_t$.
      ii. For each $i \in \{0, 1, \ldots, L\}$ and for each $j \in \{0, 1, \ldots, 2^i - 1\}$, set $\mathcal{D}_i[j] := \emptyset$.
      iii. For each element $y \in S$, let $j := H_2(y)_{[1..L]}$ and add y into the node $\mathcal{D}_L[j]$. If the size of $\mathcal{D}_L[j]$ exceeds 4σ, then abort.
      iv. For each $j \in \{0, 1, \ldots, 2^L - 1\}$, construct a node $\mathcal{D}'_L[j]$ of size 4σ by padding $\mathcal{D}_L[j]$ with dummy random elements. Compute $\widetilde{\mathcal{D}}'_L[j] \leftarrow \mathsf{Enc}_{\mathsf{pk}_1}(\mathcal{D}'_L[j])$.

   Finally, for each $L \in \{0, \ldots, \mathsf{max}L_d\}$, P_1 sends $\left\{\widetilde{\mathcal{D}}'_L[j]\right\}_{j \in \{0,1,\ldots,2^L-1\}}$ to P_0 if $\{\widetilde{\mathcal{D}}'_L[j]\} \neq \emptyset$.
4. P_0 updates $\widetilde{\mathcal{D}}$ by doing the following:
   For each $L \in \{0, \ldots, \mathsf{max}L_d\}$, $j \in \{0, 1, \ldots, 2^L - 1\}$: if P_0 received $\widetilde{\mathcal{D}}'_L[j]$ from P_1 in the above step, set $\widetilde{\mathcal{D}}_L[j] := \widetilde{\mathcal{D}}'_L[j]$; else, set $\widetilde{\mathcal{D}}_L[j] := \emptyset$.
5. P_0 learns $I_{X,\mathsf{new}} = X_d \cap Y_{[d]}$:
   P_0 first sets $I_{X,\mathsf{new}} := \emptyset$. Then for each $x \in X_d$:
   (a) P_0 does the following:
      i. Set $\mathcal{C}_0 := \emptyset$.
      ii. For each $i \in \{0, \ldots, \mathsf{max}L\}$, let $j := H_2(x)_{[1..i]}$; if $\widetilde{\mathcal{D}}_i[j] \neq \emptyset$, then for each $\mathsf{ct} \in \widetilde{\mathcal{D}}_i[j]$:
      Sample $\alpha \xleftarrow{\$} \mathbb{F}_p$, compute $\mathsf{ct}_\alpha \leftarrow \mathsf{Enc}_{\mathsf{pk}_0}(\alpha)$, $\mathsf{ct}_\beta \leftarrow \mathsf{Enc}_{\mathsf{pk}_1}(x + \alpha) \ominus \mathsf{ct}$, and add $(\mathsf{ct}_\alpha, \mathsf{ct}_\beta)$ to $\mathcal{C}_0$.
      iii. Send $\mathcal{C}_0$ to P_1.
   (b) P_1 does the following:
      i. Set $\mathcal{C}_1 := \emptyset$.
      ii. For each pair $(\mathsf{ct}_\alpha, \mathsf{ct}_\beta) \in \mathcal{C}_0$, sample $\gamma \xleftarrow{\$} \mathbb{F}_p$, compute $\beta \leftarrow \mathsf{Dec}_{\mathsf{sk}_1}(\mathsf{ct}_\beta)$ and $\mathsf{ct}_r \leftarrow \gamma \odot (\mathsf{Enc}_{\mathsf{pk}_0}(\beta) \ominus \alpha)$ and add ct_r to $\mathcal{C}_1$.
      iii. Send $\mathcal{C}_1$ in a randomly permuted order to P_0.
   (c) P_0 does the following:
      For each $\mathsf{ct}_r \in \mathcal{C}_1$, compute $r \leftarrow \mathsf{Dec}_{\mathsf{sk}_0}(\mathsf{ct}_r)$ and add x to the set $I_{X,\mathsf{new}}$ if $r = 0$.
6. P_0 computes and outputs $I_d := I_{d-1} \cup I_{X,\mathsf{old}} \cup I_{X,\mathsf{new}}$.
7. P_0 updates X_{old} and H_X as in Figure 6.
8. Finally, both parties update $d^* := d^* + N_d/\sigma$.

Figure 7: One-sided UPSI with addition protocol $\Pi_{\mathsf{UPSI\text{-}add\text{-}one}}$ when $N_d \neq \sigma$.

$\forall L \in \{0, \ldots, \mathsf{max}L_d\}$ where $\{\widetilde{\mathcal{D}}'_L[j]\} \neq \emptyset$. Naturally, this reflects only the "latest state" of any level in the tree. Finally, execute Step 7 only on sub-day $(d^* + N_d/\sigma - 1)$. For completeness, we describe the whole protocol in Figure 7. Correctness and security naturally extend from the basic protocol where $N_d = \sigma$.

5.5 Optimizations

In this section, we discuss some optimizations to improve the concrete communication and computational efficiency of the protocol.

Cuckoo hashing. In Step 3(b)iii, for each element $y \in S$, instead of adding y to the (end of) node $D_L[j]$, we store elements in each node using Cuckoo hashing [PR04]. In more detail, to implement Cuckoo hashing, as discussed in Section 2, we pick three hash functions $\mathsf{CuH}_1, \mathsf{CuH}_2, \mathsf{CuH}_3$. Each node of the tree $D_L[j]$ is represented as a collection of b bins. We also have a small stash associated with each node. Now, each y is inserted into one of these b bins (or the stash) at any given node depending on the contents of bins $\mathsf{CuH}_1(y), \mathsf{CuH}_2(y), \mathsf{CuH}_3(y)$. Similarly, we also include the elements from the stash when defining S and setting $\mathcal{D}_i[j] = \emptyset$ in Step 3b.

The advantage is that, in Step 5, for each $x \in X_d$, $i \in \{0, \ldots, \mathsf{max}L\}$, non-empty node $\widetilde{\mathcal{D}}_i[j]$ (where $j = H_2(x)_{[1,\ldots,i]}$), instead of comparing x with each of the 4σ elements in the node, P_0 needs to compare with only the three elements at bins $\mathsf{CuH}_1(x), \mathsf{CuH}_2(x), \mathsf{CuH}_3(x)$ and those in the associated stash. This significantly reduces the communication and computation cost.

In our implementation, we set the Cuckoo hashing parameters according to the work of Pinkas et al. [PSSZ15]. In particular, we can set the number of bins $b = 5\sigma$ and stash size to be a small constant. See Section 7.3 for more details.

El Gamal encryption. We instantiate the additively homomorphic encryption scheme using the exponential variant of the El Gamal scheme [Gam84] to take advantage of the efficient elliptic curve cryptographic operations. Recall that in this scheme, $\mathsf{Enc}(m) = (g^r, h^r \cdot g^m)$ where the public key consists of a generator g and group element $h = g^x$. The secret key is x. In our protocol, let $\mathsf{pk}_0 = (g, h_0), \mathsf{pk}_1 = (g, h_1), \mathsf{sk}_0 = x_0, \mathsf{sk}_1 = x_1$ – that is, both parties use the same group and generator g. First, in Step 5c, instead of decrypting ct_r entirely,[3] P_0 can just check if the decryption is 0 more efficiently. In particular, given $\mathsf{ct}_r = (a, b)$, P_0 can check if $r = 0$ by checking if $b = a^{x_0}$. Similarly, in Step 5(b)ii, given $\mathsf{ct}_\beta = (a, b)$, instead of decrypting to get β and then re-encrypting using pk_0, P_1 can compute $\mathsf{Enc}_{\mathsf{pk}_0}(\beta)$ directly as $(g^s, h_0^s \cdot \frac{b}{a^{x_1}})$ where s is randomly sampled.

Reducing number of ciphertexts in $\mathcal{C}_0$. We can reduce communication by modifying Step 5(a)ii to allow P_0 to use the same ct_α across all the ciphertext tuples generated for a given $x \in X_d$. In more detail, we rewrite the step as:

- Sample $\alpha \xleftarrow{\$} \mathbb{F}_p$. Compute $\mathsf{ct}_\alpha \leftarrow \mathsf{Enc}_{\mathsf{pk}_0}(\alpha)$ and add ct_α to $\mathcal{C}_0$.
- For each $i \in \{0, \ldots, \mathsf{max}L\}$, let $j := H_2(x)_{[1..i]}$; if $\widetilde{\mathcal{D}}_i[j] \neq \emptyset$, then for each $\mathsf{ct} \in \widetilde{\mathcal{D}}_i[j]$: compute ct_β as $\big(\mathsf{Enc}_{\mathsf{pk}_1}(\alpha) \oplus \beta_r \odot (\mathsf{Enc}_{\mathsf{pk}_1}(x) \ominus \mathsf{ct})\big)$ where $\beta_r \xleftarrow{\$} \mathbb{F}_p$ and $(\oplus, \ominus, \odot)$ are homomorphic operations. That is, $\mathsf{ct}_\beta = \mathsf{Enc}_{\mathsf{pk}_1}\big(\alpha + \beta_r \cdot (x - y)\big)$ where $\mathsf{ct} = \mathsf{Enc}_{\mathsf{pk}_1}(y)$.

[3] Decryption of exponential variant of El Gamal requires computing the discrete logarithm of a group element which would only work for a small message space and be expensive.

- Add ct_β to $\mathcal{C}_0$.

This change does not leak any additional information to P_1 because, by assumption, since elements added by P_1 on any day are distinct, with all but negligible probability, $x = y$ for at most only one y amongst the plaintexts encrypted to form ciphertexts $\{\mathsf{ct}\}$ (the negligible probability error happens if x equals any of the random dummy elements too). For any $x \neq y$, $\beta_r \cdot (x - y)$ is statistically close to a uniform distribution since β_r is picked uniformly at random and hence reveals no information about any x to P_1. It is easy to observe that this optimization does not affect security against a corrupt P_0 as well. This optimization reduces the size of $\mathcal{C}_0$ by half.

5.6 Efficiency

In this section, we evaluate the communication and computational complexity of the protocol (after applying the optimizations). For simplicity, we analyze the case where $N_d = \sigma$. Recall the notation $\mathsf{Num1}(n)$ that denotes the number of 1's in the binary representation of n. For any Day d, $\mathsf{Num1}(d)$ is the number of levels of the tree that are non-empty. Let the stash size (a small constant) for any node in the tree be denoted by s (which is a small constant). Over a period of d days, the total number of elements in the input set of each party is $N = \sigma \cdot d$.

Communication Complexity. In Step 1, P_1 sends σ group elements. In Step 3, P_1 sends $(2^L \cdot 5\sigma)$ ciphertexts, where $L = \mathsf{LS1}(d)$. In Step 5, P_0 first sends $\sigma \cdot (1 + \mathsf{Num1}(d) \cdot (s + 3))$ ciphertexts and P_1 responds back with $\sigma \cdot (\mathsf{Num1}(d) \cdot (s + 3))$ ciphertexts. In Step 7, both parties send σ group elements. Thus, the overall communication complexity is $O(\sigma \cdot (2^{\mathsf{LS1}(d)} + \mathsf{Num1}(d)))$ group elements.

Now, the values of $\mathsf{LS1}(d)$ and $\mathsf{Num1}(d)$ differ on every day and so the communication cost is not the same on each day. We consider amortized cost over 2^k days of updates for $d \in \{2^k, 2^k + 1, \ldots, 2^{k+1} - 1\}$. The amortized $2^{\mathsf{LS1}(d)}$ is

$$\frac{\sum_{d=2^k}^{2^{k+1}-1} 2^{\mathsf{LS1}(d)}}{2^k} = \frac{2^k + \sum_{i=0}^{k-1} 2^i \cdot 2^{k-1-i}}{2^k} = 1 + \frac{k}{2}.$$

The amortized $\mathsf{Num1}(d)$ is

$$\frac{\sum_{d=2^k}^{2^{k+1}-1} \mathsf{Num1}(d)}{2^k} = \frac{2^k + k \cdot 2^{k-1}}{2^k} = 1 + \frac{k}{2}.$$

Hence, the amortized communication cost over 2^k days is $O(\sigma \cdot k)$. Since the total number of elements $N = \sigma \cdot d$, we know that $k = O(\log N)$ and so the amortized communication cost is $O(\sigma \cdot \log N)$. In particular, it grows only logarithmically with the total number of elements.

Computational Complexity. First, we analyze the computation cost for P_0. In Step 1, P_0 performs σ exponentiations. In Step 4, P_0 stores the 2^L nodes – this is inexpensive compared to exponentiations. In Step 5, P_0 generates $\sigma \cdot (1 + \mathsf{Num1}(d) \cdot (s + 3))$ ciphertexts (and decrypting later to check for 0). In Step 7, P_0 does 2σ exponentiations. Hence P_0's computation cost is $O(\sigma \cdot \mathsf{Num1}(d))$.

Next, we analyze P_1's cost. In Step 1, P_1 does σ exponentiations. In Step 3, P_1 generates $(2^L \cdot 5\sigma)$ encryptions, where $L = \mathsf{LS1}(d)$. In Step 5, P_1 performs $\sigma \cdot (1 + \mathsf{Num1}(d) \cdot (s + 3))$

encryptions/homomorphic evaluations. In Step 7, P_1 does σ exponentiations. So P_1's computation cost is $O(\sigma \cdot (2^{\mathsf{LS1}(d)} + \mathsf{Num1}(d)))$.

As analyzed above, the amortized computation cost is $O(\sigma \cdot k)$ over 2^k days of updates for $d \in \{2^k, 2^k + 1, \ldots, 2^{k+1} - 1\}$. Since the total number of elements $N = \sigma \cdot d$, we have $k = O(\log N)$ and the amortized computation cost is $O(\sigma \cdot \log N)$, which grows only logarithmically with the total number of elements.

Discussion for $N_d \neq \sigma$. In our protocol, intuitively, since we run $\lceil N_d/\sigma \rceil$ instances of the basic case (where inputs are of size σ), an upper bound on the communication and computation cost on any day is $\lceil N_d/\sigma \rceil$ times that of the basic case's cost. We can in fact do better than just repeating the protocol so many times but we ignore that for the sake of simplifying the analysis and provide a relatively loose upper bound. In conclusion, the amortized communication and computation cost (for each party) is $O(\lceil N_d/\sigma \rceil \cdot \sigma \cdot \log N)$. Once again, this grows only logarithmically with the total number of elements so far.

6 Updatable PSI with Weak Deletion

In this section, we describe an updatable PSI protocol satisfying Definition 3.3. That is, besides inserting new elements to their sets each day, the protocol allows both parties to delete data that was added t days ago and compute the intersection privately on these new updated sets. In particular, the output is the union of the intersection of each party's new elements with the other party's updated set comprising elements over the last t days. Our protocol allows both parties to learn the output at the end of each day and is based on oblivious transfer (OT) and correlation robust hash functions.

Next, we first introduce the notion of sender-streaming PSI and then use that to build our updatable PSI protocol with weak deletion.

6.1 Sender-streaming PSI

Consider two parties - a sender S and a receiver R who wish to engage in a one-sided PSI protocol to allow R to learn the intersection without revealing anything else. However, unlike the typical PSI setting, only R knows its entire input set Y at the beginning while the sender only knows a subset X_0. An upper bound Max on the maximum number of elements in the sender's set is part of the public parameters as are the sizes $|Y|, |X_0|$. At this point, the receiver learns $(X_0 \cap Y)$. Subsequently, the sender learns more of its input in a streaming manner and the two parties interact to allow the receiver to learn the intersection of its input set with the new streamed sender input. That is, on receiving an streaming input X_i, the two parties engage in a protocol that allows the receiver to learn $(X_i \cap Y)$. We formalize this notion as a special case of secure two-party computation with a reactive functionality defined in Figure 8.

Let $X_{[i]} = \{X_0, \ldots, X_i\}$ be the inputs of S over i streams and Y be the input of R. Let $\mathsf{View}_S^{\Pi,i}(X_{[i]}, Y, \mathsf{Max})$, $\mathsf{View}_R^{\Pi,i}(X_{[i]}, Y, \mathsf{Max})$ be the views of S and R, respectively, in the protocol Π at the end of i streams and let $\mathsf{Out}^{\Pi,i}(X_{[i]}, Y, \mathsf{Max})$ be the outputs of R at the end of i streams. Let $f(X_{[i]}, Y, \mathsf{Max}) := \{I_0, \ldots, I_i\}$ be the outputs of the ideal functionality in the i streams.

Definition 6.1. ***(Sender-streaming PSI.)*** *A protocol Π is semi-honest secure with respect to ideal functionality $\mathcal{F}_{\mathsf{SSPSI}}$ if there exists PPT simulators Sim_S and Sim_R such that for any $i \in \mathbb{N}$,*

Initialization:

- **Inputs:**
  S inputs a set X_0 where each element is from $\{0,1\}^*$.
  R inputs a set Y where each element is from $\{0,1\}^*$.
  The set sizes $|X_0|, |Y|$ and upper bound Max are public and known to both parties.
- **Output:**
  The ideal functionality sets $X = X_0$. Then, it computes and sends $X_0 \cap Y$ to R.

Stream i:

- **Inputs:**
  S inputs a set X_i where each element is from $\{0,1\}^*$ and $X_i \cap X = \emptyset$. The stream size $|X_i|$ is public and known to R.
- **Output:**
  The ideal functionality sets $X = X \cup X_i$. Then, if $|X| \leq \mathsf{Max}$, it computes and sends $I_i = X_i \cap Y$ to R. Else, sends $\perp$.

Figure 8: Ideal functionalities $\mathcal{F}_{\mathsf{SSPSI}}$ for sender-streaming PSI.

any inputs $(X_{[i]}, Y)$ *and any upper bound* Max,

$$\mathsf{View}_R^{\Pi,i}(X_{[i]}, Y, \mathsf{Max}) \overset{c}{\approx} \mathsf{Sim}_R\left(1^\lambda, Y, \{|X_j|\}_{j\in[i]}, \mathsf{Max}, f(X_{[i]}, Y, \mathsf{Max})\right),$$
$$\left(\mathsf{View}_S^{\Pi,i}(X_{[i]}, Y, \mathsf{Max}), \mathsf{Out}^{\Pi,i}(X_{[i]}, Y, \mathsf{Max})\right) \overset{c}{\approx} \left(\mathsf{Sim}_S\left(1^\lambda, X_{[i]}, |Y|, \mathsf{Max}\right), f(X_{[i]}, Y, \mathsf{Max})\right).$$

6.1.1 Instantiations

We now informally describe how the PSI protocols of Kolesnikov et al. [KKRT16], Pinkas et al. [PRTY19], Chase and Miao [CM20] immediately satisfy Definition 6.1. Each of these protocols is based on semi-honest OT and correlation robust hash functions. At a high level, in each of these protocols, the sender's input is used only in the last step to compute the oblivious pseudo-random function (OPRF) values before they are sent to the receiver. As a result, this can be done in a streaming manner so long as the maximum number of values to be computed upon are known apriori to set up the OPRF key. We now provide more details.

Protocol Structure. All the three protocols have the following high level structure. Consider a sender S with input set X and receiver R with input set Y. In the first phase, both parties run an interactive protocol to jointly generate a key K for an OPRF and evaluate R's input on this OPRF obliviously. In a bit more detail, at the end of this interactive protocol, S learns the key K (and nothing about R's input) and R learns the evaluations $\{\mathsf{OPRF}(K, y)\}_{y\in Y}$ (and nothing about the key K). For each $z = \mathsf{OPRF}(K, y)$, the receiver also learns that this is the evaluation of its corresponding input element y (that is, the outputs aren't permuted). In the protocol of Kolesnikov et al. [KKRT16], R's inputs are first separated into various buckets via Cuckoo hashing [PR04] and a separate instance of this OPRF protocol is run for each bucket - the sender learns one OPRF key K_i for each bucket and R learns $\mathsf{OPRF}(K_i, y)$ for the element y that falls into this bucket. In the protocols of [PRTY19] and [CM20], a multi-point OPRF is set up where S learns a single key

K and R learns the evaluation of all its points - $\{\mathsf{OPRF}(K, y)\}_{y \in Y}$. Our key observation is that, crucially, so far, the sender does not need to know its input. Instead, S only needs to know the size (or an upper bound) of its input set to allow the key K to be chosen.

In the next phase, S sends evaluations of the OPRF on its input elements to R. That is, S, now in possession of key K locally computes and sends $\{\mathsf{OPRF}(K, x)\}_{x \in X}$ (in the case of [KKRT16], S evaluates $\mathsf{OPRF}(K_i, x)$ for every possible $x \in X$ that can fall into bucket i, for each i). The receiver can then compare $\{\mathsf{OPRF}(K, x)\}_{x \in X}$ with $\{\mathsf{OPRF}(K, y)\}_{y \in Y}$ to compute the intersection. The security guarantee of the OPRF is that for any $x \in X \setminus I$, $\mathsf{OPRF}(K, x)$ appears pseudorandom to R and hence leaks no information about the element x.

Now, observe that in our setting of sender-streaming PSI, the two parties can run the first phase with the sender only providing an upper bound of the number of elements it will eventually stream. Then, for any stream X_i (including the initial one X_0), S can compute and send $\{\mathsf{OPRF}(K, x)\}_{x \in X_i}$ and R can then immediately learn $(X_i \cap Y)$. The security of this protocol immediately follows from that of the underlying PSI protocol so long as the number of elements streamed by the sender is less than the upper bound that was set. As a result, we get the following lemma:

Lemma 6.2. *Assuming semi-honest OT and correlation robust hash functions, the PSI protocols of [KKRT16, PRTY19, CM20] all securely realize the ideal functionality $\mathcal{F}_{\mathsf{SSPSI}}$ against semi-honest adversaries.*

Efficiency. We now briefly analyze the communication and computation cost of realizing $\mathcal{F}_{\mathsf{SSPSI}}$ using each of these three instantiations [KKRT16, PRTY19, CM20]. In the initialization phase, to set up the OPRF key (and evaluate R's input), the computational complexity (per party) is $O(|Y| \cdot \lambda)$ and the communication complexity (from R to S) is $O(|Y| \cdot \lambda)$ bits, where λ is the security parameter. Note that the communication also grows with Max, but the growth is dominated by $O(|Y| \cdot \lambda)$ for any polynomially large Max, so we omit it here. Then for each stream X_i (including X_0), S evaluates and sends the OPRF values on X_i, where the computational complexity (of S) is $O(|X_i| \cdot \lambda)$ and the communication complexity (from S to R) is $O(|X_i| \cdot \sigma)$, where σ is the statistical security parameter.

6.2 Construction

Notation. On each Day d, let X_d be the elements added to P_0's set and Y_d be added to P_1's set where $|X_d| = |Y_d| = N_d$. For any j, we will initialize $X_j = Y_j = \emptyset$ if they have not yet been defined (or $j \leq 0$). Further, for any d, let $\mathsf{Max}_d \geq (2 \cdot N_d + \sum_{j=d+1}^{j=d+t-1} N_j)$. We assume that Max_d is known at the start of Day d - that is, on any day, both parties know an upper bound on the number of elements they can add over the next $(t-1)$ days.

Since we invoke several instances of $\mathcal{F}_{\mathsf{SSPSI}}$ in our protocol, we introduce additional notation to identify the sender of $\mathcal{F}_{\mathsf{SSPSI}}$ and on which day of the UPSI protocol the functionality was first invoked. Let $\mathcal{F}_{\mathsf{SSPSI}}^{(P_0,d)}$ indicate that P_0 is the sender of the SS-PSI protocol, P_1 is the receiver and the functionality was first invoked on Day d.

Construction Overview. We focus on how P_0 computes the output - the final protocol is symmetric to allow P_1 to compute the output as well. On any Day d, observer that the output I_d can be split into two disjoint sets: (i) $I_{0,\alpha} = (X_{[d-1]} \setminus X_{[d-t]}) \cap Y_d$ and (ii) $I_{0,\beta} = X_d \cap (Y_{[d]} \setminus Y_{[d-t]})$.

Then, $I_d = I_{0,\alpha} \dot{\cup} I_{0,\beta}$ (each of which can be inferred from the output in the ideal world). Note that $X_d \cap Y_d$ is included in $I_{0,\beta}$ and not $I_{0,\alpha}$.

To compute $I_{0,\alpha}$, note that $I_{0,\alpha} = (X_{d-t+1} \cap Y_d) \dot{\cup} \ldots \dot{\cup} (X_{d-1} \cap Y_d)$, where $(X_j \cap Y_d)$ (for all $j \in \{d-t+1, \ldots, d-1\}$) can be inferred from the output in the ideal word. Our idea is to use the sender-streaming PSI ($\mathcal{F}_{\mathsf{SSPSI}}$) initiated on earlier days to let P_0 learn $(X_j \cap Y_d)$. In more detail, on each of the $(t-1)$ previous days, invoke $\mathcal{F}_{\mathsf{SSPSI}}^{(P_1,j)}$ with P_0 as receiver using input X_j (on Day j) and P_1 as sender. The upper bound for the sender's set size is discussed later. Then, on Day d, P_1's streamed input for each of these instances is Y_d which allows P_0 to learn $(X_j \cap Y_d)$. The same mechanism can be employed symmetrically for P_1 to learn $I_{1,\alpha} = X_d \cap (Y_{[d-1]} \setminus Y_{[d-t]})$.

Next, to compute $I_{0,\beta}$, the idea is to use a new instance $\mathcal{F}_{\mathsf{SSPSI}}^{(P_1,d)}$ on Day d with P_0 as the receiver using input X_d and P_1 as the sender. From the above paragraph, observe that this instance of $\mathcal{F}_{\mathsf{SSPSI}}$ is also used to compute terms of $I_{0,\alpha}$ over the following $(t-1)$ days. Now, since the goal is to compute $I_{0,\beta} = X_d \cap (Y_{[d]} \setminus Y_{[d-t]})$, sender P_1's input in its initial stream should be $(Y_{[d]} \setminus Y_{[d-t]})$ whose size is $\sum_{j=d-t+1}^{j=d} N_j$. Nonetheless, this can be improved. Observe that $I_{0,\beta} = (X_d \cap Y_d) \cup \Big(X_d \cap (Y_{[d-1]} \setminus Y_{[d-t]})\Big) = X_d \cap (Y_d \cup I_{1,\alpha})$. Thus, sender P_1's input to $\mathcal{F}_{\mathsf{SSPSI}}^{(P_1,d)}$ can be just $(Y_d \cup I_{1,\alpha})$. Since $|I_{1,\alpha}| \leq |N_d|$, size of P_1's input is at most $(2 \cdot N_d)$. P_1 uses dummy elements to pad the size to be exactly $2 \cdot N_d$ to not leak more information about $I_{1,\alpha}$ to P_0. Once again, P_1 can similarly learn $I_{1,\beta} = Y_d \cap (X_{[d]} \setminus X_{[d-t]})$.

Finally, the missing component is an upper bound on sender P_1's entire input in $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,d}$ initiated on Day d. Recall that to compute $I_{0,\alpha}$, for each of the next $(t-1)$ days, P_1 uses streamed input Y_j on Day j. Hence the upper bound is $(2 \cdot N_d + \sum_{j=d+1}^{j=d+t-1} N_j)$. The protocol is described in Figure 9.

6.3 Correctness and Efficiency

Correctness. If both parties follow the protocol honestly, at the end of Day d, we will have the guarantee that with all but negligible probability, $I_d = \big((X_{[d]} \setminus X_{[d-t]}) \cap Y_d\big) \cup \big((Y_{[d]} \setminus Y_{[d-t]}) \cap X_d\big)$.

We prove this by induction. It is easy to observe that this guarantee holds on Day 1 since, by the correctness of the initialization phase of $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,1}$ and $\mathcal{F}_{\mathsf{SSPSI}}^{P_0,1}$, both parties learn the intersection $(X_1 \cap Y_1)$.

Now consider Day d with new input sets X_d an Y_d respectively. In Step 1, for each $j > 0$, by the correctness of the j^{th} stream of $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,j}$, P_0 indeed learns $X_j \cap Y_d$. Thus,

$$\begin{aligned} I_{0,\alpha} &= \bigcup_{i=(d-t+1)}^{d-1} (X_i \cap Y_d) \\ &= (X_{[d-1]} \setminus X_{[d-t]}) \cap Y_d. \end{aligned}$$

Similarly, in Step 2, for each $j > 0$, P_1 indeed learns $Y_j \cap X_d$. Thus, $I_{1,\alpha} = (Y_{[d-1]} \setminus Y_{[d-t]}) \cap X_d$.

In Step 3, by the correctness of the initialization phase of $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,d}$, P_0 learns

$$\begin{aligned} I_{0,\beta} &= (X_d \cap B) \\ &= (X_d \cap Y_d) \cup \Big(\bigcup_{i=d-t+1}^{d-1} Y_i \cap X_d \cap X_d \Big) \cup (X_d \cap \widehat{D_Y}) \end{aligned}$$

Day 1: P_0 has input set X_1 and P_1 has input set Y_1. The protocol works as follows:

1. Invoke $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,1}$ with P_0 as the receiver with input X_1, P_1 as the sender with initial input Y_1 and upper bound Max_1. P_0 learns output $I_1 = (X_1 \cap Y_1)$.

2. Invoke $\mathcal{F}_{\mathsf{SSPSI}}^{P_0,1}$ with P_1 as the receiver with input Y_1, P_0 as the sender with initial input X_1 and upper bound Max_1. P_1 learns output $I_1 = (X_1 \cap Y_1)$.

Day d**:** P_0 has new input set X_d and P_1 has new input set Y_d. P_0's and P_1's input sets over the last t days are $\big(X_{d-t+1}, \ldots, X_{d-1}, X_d\big)$ and $\big(Y_{d-t+1}, \ldots, Y_{d-1}, Y_d\big)$ respectively. The protocol works as follows:

1. $I_{0,\alpha} = \bigcup_{j=(d-t+1)}^{d-1}(X_j \cap Y_d)$: For each $j > 0$, invoke $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,j}$ with P_1's new streamed input as Y_d. Receiver P_0 learns $(X_j \cap Y_d)$.

2. $I_{1,\alpha} = \bigcup_{j=(d-t+1)}^{d-1}(Y_j \cap X_d)$: For each $j > 0$, invoke $\mathcal{F}_{\mathsf{SSPSI}}^{P_0,j}$ with P_0's new streamed input as X_d. Receiver P_1 learns $(Y_j \cap X_d)$.

3. $I_{0,\beta} = \Big(X_d \cap \big(\bigcup_{j=d-t+1}^{d} Y_j\big)\Big)$: P_0 computes this as follows:

   (a) Invoke $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,d}$ with P_0 as the receiver with input X_d, P_1 as the sender with initial input B and upper bound Max_d where the set $B = Y_d \cup \big(\bigcup_{j=d-t+1}^{d-1} Y_j \cap X_d\big) \cup \widehat{D_Y}$ where $\widehat{D_Y}$ consists of dummy random elements so that $|B| = 2 \cdot N_d$.

   (b) P_0's output is $\Big(X_d \cap \big(\bigcup_{j=d-t+1}^{d} Y_j\big)\Big)$ since $\Big(X_d \cap \big(\bigcup_{j=d-t+1}^{d} Y_j\big)\Big) = \big(X_d \cap B\big)$.

4. $I_{1,\beta} = \Big(Y_d \cap \big(\bigcup_{j=d-t+1}^{d} X_j\big)\Big)$: P_1 computes this similar to the above as follows:

   (a) Invoke $\mathcal{F}_{\mathsf{SSPSI}}^{P_0,d}$ with P_1 as the receiver with input Y_d, P_0 as the sender with initial input A and upper bound Max_d where the set $A = X_d \cup \big(\bigcup_{j=d-t+1}^{d-1} X_j \cap Y_d\big) \cup \widehat{D_X}$ where $\widehat{D_X}$ consists of dummy random elements so that $|A| = 2 \cdot N_d$.

   (b) P_1's output is $\Big(Y_d \cap \big(\bigcup_{j=d-t+1}^{d} X_j\big)\Big)$ since $\Big(Y_d \cap \big(\bigcup_{j=d-t+1}^{d} X_j\big)\Big) = \big(Y_d \cap A\big)$.

5. **Output computation:**
   P_0 outputs $I_d = (I_{0,\alpha} \cup I_{0,\beta})$ and P_1 outputs $I_d = (I_{1,\alpha} \cup I_{1,\beta})$.

Figure 9: Updatable PSI protocol with weak deletion $\Pi_{\mathsf{UPSI\text{-}del}}$.

$$
\begin{aligned}
&= (X_d \cap Y_d) \cup (X_d \cap (Y_{[d-1]} \setminus Y_{[d-t]})) \cup \emptyset \qquad \text{(with overwhelming prob. as } \widehat{D_Y} \text{ is random)} \\
&= X_d \cap (Y_{[d]} \setminus Y_{[d-t]})
\end{aligned}
$$

Similarly, in Step 4, $I_{1,\beta} = Y_d \cap (X_{[d]} \setminus X_{[d-t]})$.

Finally, P_0 outputs:

$$\begin{aligned} I_d &= I_{0,\alpha} \cup I_{0,\beta} \\ &= \Big((X_{[d-1]} \setminus X_{[d-t]}) \cap Y_d\Big) \cup \Big(X_d \cap (Y_{[d]} \setminus Y_{[d-t]})\Big) \\ &= \Big((X_{[d-1]} \setminus X_{[d-t]}) \cap Y_d\Big) \cup \Big(X_d \cap Y_d\Big) \cup \Big(X_d \cap (Y_{[d-1]} \setminus Y_{[d-t]})\Big) \\ &= \Big((X_{[d]} \setminus X_{[d-t]}) \cap Y_d\Big) \cup \Big(X_d \cap (Y_{[d]} \setminus Y_{[d-t]})\Big) \end{aligned}$$

Similarly, we can prove that P_1 also outputs the same.

Computational and Communication Complexity. On Day d, $\mathcal{F}_{\mathsf{SSPSI}}$ is invoked with a new stream $2 \cdot (t-1)$ times with size of new streamed set as N_d. The total computational complexity in this step is $O(N_d \cdot \lambda \cdot t)$ and the communication complexity is $O(N_d \cdot \sigma \cdot t)$ bits. Besides, two new invocations of $\mathcal{F}_{\mathsf{SSPSI}}$ (the initialization phase) occur where the receiver's set size is N_d and the sender's set size is $2 \cdot N_d$. The computational complexity in this step is $O(N_d \cdot \lambda)$ and communication complexity is $O(N_d \cdot \lambda)$ bits. Thus, the total computational complexity is $O(N_d \cdot \lambda \cdot t)$ and the total communication complexity is $O(N_d \cdot (\sigma \cdot t + \lambda))$ bits.

6.4 Security

Theorem 6.3. *The protocol $\Pi_{\mathsf{UPSI\text{-}del}}$ presented in Figure 9 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$ (defined in Figure 3) in the $\mathcal{F}_{\mathsf{SSPSI}}$-hybrid model against semi-honest adversaries.*

Instantiating $\mathcal{F}_{\mathsf{SSPSI}}$ with the protocol of Kolesnikov et al. [KKRT16] or Pinkas et al. [PRTY19] or Chase and Miao [CM20], all of which are based on semi-honest OT and correlation robust hash functions (Lemma 6.2), we get the following corollary:

Corollary 6.4. *Assuming semi-honest OT and correlation robust hash functions, the protocol $\Pi_{\mathsf{UPSI\text{-}del}}$ presented in Figure 9 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$ (defined in Figure 3) against semi-honest adversaries.*

Security against corrupted P_0. Consider an adversary $\mathcal{A}$ that corrupts party P_0. We construct a PPT Sim_0 that, on input $\big(1^\lambda, X_{[D]}, N_{[D]}, f(X_{[D]}, Y_{[D]})\big)$, where $f(X_{[D]}, Y_{[D]}) := \{I_1, \ldots, I_D\}$ are the outputs of the ideal functionality in the D days, interacts with adversary $\mathcal{A}$ as follows and outputs $\mathcal{A}$'s view.

Day 1: On behalf of functionality $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,1}$, send output $I_1 = f(X_{[1]}, Y_{[1]})$ to $\mathcal{A}$.

Day d:

1. $\underline{I_{0,\alpha} = \bigcup_{j=(d-t+1)}^{d-1} (X_j \cap Y_d)}$: For each $j > 0$, on behalf of functionality $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,j}$, send $\big(X_j \cap (I_d \setminus I_{d-1})\big)$ to $\mathcal{A}$. Observe that this is equal to $X_j \cap Y_d$ since X_i's are mutually disjoint sets.
2. $\underline{I_{0,\beta} = \Big(X_d \cap \big(\bigcup_{j=d-t+1}^{d} Y_j\big)\Big)}$: Observe that $(X_d \cap I_d) = \big(X_d \cap (Y_{[d]} \setminus Y_{[d-t]})\big)$. Thus, on behalf of functionality $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,d}$, send $(X_d \cap I_d)$ to $\mathcal{A}$.

We now show that the above simulation strategy against a corrupt P_0 is successful via a series of hybrid arguments where Hyb_0 corresponds to the real world and Hyb_3 corresponds to the ideal world execution.

1. Hyb_0: This corresponds to the real world execution where $\mathcal{A}$ interacts with a simulator SimHyb that plays the role of honest P_1.
2. Hyb_1: In this hybrid, on Day 1, SimHyb sends output I_1 to $\mathcal{A}$ on behalf of the ideal functionality $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,1}$. This is part of Sim_0's input on Day 1 of the protocol.
3. Hyb_2: In this hybrid, on any Day d, to compute the term $I_{0,\alpha}$, for each $j \in \{d-t+1, \ldots, d-1\}$, $j > 0$, on behalf of functionality $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,j}$, SimHyb sends $\big(X_j \cap (I_d \setminus I_{d-1})\big)$ to $\mathcal{A}$, where X_j, I_d, I_{d-1} are part of Sim_0's input.
4. Hyb_3: In this hybrid, on any Day d, to compute the term $I_{0,\beta}$, on behalf of functionality $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,d}$, SimHyb sends $(X_d \cap I_d)$ to $\mathcal{A}$.

We now show that every pair of successive hybrids is computationally indistinguishable.

Lemma 6.5. Hyb_0 *is identically distributed to* Hyb_1.

Proof. In both hybrids, $\mathcal{A}$ sends input X_1 to $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,1}$. In Hyb_0, honest P_1 sends input (Y_1, Max_1) to $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,1}$. By the correctness of $\mathcal{F}_{\mathsf{SSPSI}}$, $\mathcal{A}$ receives output $(X_1 \cap Y_1)$. In Hyb_1, by the definition of functionality $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$, the value $I_1 = f(X_1, Y_1)$ sent by SimHyb on behalf of $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,1}$ is equal to $(X_1 \cap Y_1)$. Since there is no other difference between the two hybrids, they are identical. □

Lemma 6.6. Hyb_1 *is identically distributed to* Hyb_2.

Proof. In Hyb_1, to compute the term $I_{0,\alpha}$, for each $j \in \{d-t+1, \ldots, d-1\}$, $j > 0$, honest P_1 sends streamed input Y_d to $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,j}$ and $\mathcal{A}$ gets output $(X_j \cap Y_d)$ by the correctness of functionality $\mathcal{F}_{\mathsf{SSPSI}}$. In Hyb_2, for each j, SimHyb sends $\big(X_j \cap (I_d \setminus I_{d-1})\big)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,j}$. By the definition of functionality $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$ and the fact that each party's input set is mutually disjoint on each day, $\big(X_j \cap (I_d \setminus I_{d-1})\big)$ is indeed equal to $(X_j \cap Y_d)$. □

Lemma 6.7. Hyb_2 *is statistically indistinguishable from* Hyb_3.

Proof. In both hybrids, to compute the term $I_{0,\beta}$, $\mathcal{A}$ sends input X_d to $\mathcal{F}_{\mathsf{SSPSI}}^{P_1,d}$. In Hyb_2, honest P_1 sends initial input (B, Max_d) where $B = Y_d \cup \big(\bigcup_{j=d-t+1}^{d-1} Y_j \cap X_d\big) \cup \widehat{D_Y}$ where $\widehat{D_Y}$ consists of dummy random elements. By the correctness of functionality $\mathcal{F}_{\mathsf{SSPSI}}$, except with negligible probability, $\mathcal{A}$ gets output $X_d \cap (Y_{[d]} \setminus Y_{[d-t]})$ - note that the only scenario when this is not the output is if $\widehat{D_Y} \cap X_d \neq \emptyset$. In this case, the output has more elements but since $\widehat{D_Y}$ consists of dummy random elements, this occurs only with negligible probability.

In Hyb_3, SimHyb sends $(X_d \cap I_d)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{SSPSI}}$. By the definition of functionality $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$, observe that $(X_d \cap I_d)$ is indeed equal to $X_d \cap (Y_{[d]} \setminus Y_{[d-t]})$. Since there is no other difference between the two hybrids, they are statistically indistinguishable. □

Security against corrupted P_1. Since the protocol is symmetric, the proof is identical to the above case where P_0 was corrupt.

6.5 Discussion

In this section, we discuss the choice of our definition for the weak deletion functionality. We consider an alternate, arguably more natural functionality for deletion where both parties compute the intersection of their datasets over the last t days - that is, delete data that was added more than t days ago and compute the intersection on their updated sets. We define this functionality $\mathcal{F}_{\mathsf{UPSI\text{-}del\text{-}alt}}$ in Figure 10. We make two observations about $\mathcal{F}_{\mathsf{UPSI\text{-}del\text{-}alt}}$ to explain why we instead choose to focus on $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$ in this work.

Initialization: $X := \emptyset, Y := \emptyset$.

Day d**:**

- **Public parameter:** The set size on Day d is N_d.
- **Inputs:**
  P_0 inputs a set X_d of size N_d where each element is from $\{0,1\}^*$, and $X_d \cap X = \emptyset$.
  P_1 inputs a set Y_d of size N_d where each element is from $\{0,1\}^*$, and $Y_d \cap Y = \emptyset$.
- **Update:** On receiving the inputs from the two parties, the ideal functionality updates $X := (X \cup X_d) \setminus X_{d-t}, Y := (Y \cup Y_d) \setminus Y_{d-t}$ and computes $I_d = X \cap Y$. (If $d - t \leq 0$, let $X_{d-t} = Y_{d-t} = \emptyset$.)
- **Output:** The ideal functionality sends I_d to both parties.

Figure 10: Ideal functionality $\mathcal{F}_{\mathsf{UPSI\text{-}del\text{-}alt}}$ for UPSI with weak deletion.

Leakage from ideal functionality. It turns out that $\mathcal{F}_{\mathsf{UPSI\text{-}del\text{-}alt}}$ in fact leaks a lot more information over the course of several days than what immediately meets the eye from the functionality description. In particular, both parties actually learn $(X_i \cap Y_j)$ for all $|i - j| < t$. To see why, consider the sequence of t days starting on Day d and see what P_0 learns from the output on each day about X_d. On Day d, P_0 can learn $(X_d \cap (Y_{[d]} \setminus Y_{[d-t]}))$ from I_d. On Day $(d+1)$, P_0 can infer $(X_d \cap (Y_{[d+1]} \setminus Y_{[d-t+1]}))$ from I_{d+1}. From both the above, P_0 can immediately deduce $(X_d \cap Y_{d+1})$ and $(X_d \cap Y_{d-t+1})$. Similarly, for each $i \in \{d+2, \ldots, d+t-1\}$, P_0 can learn $(X_d \cap Y_i)$ and $(X_d \cap Y_{i-t})$. Finally, notice that on Day $(d+t-1)$, P_0 learns $X_d \cap (Y_{[d+t-1]} \setminus Y_{[d-1]})$. From this and the intermediate results on each day, P_0 can also learn $X_d \cap Y_d$. Observe that this leakage does not occur in $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$.

Stronger functionality ($\mathcal{F}_{\mathsf{UPSI\text{-}del}} \Rightarrow \mathcal{F}_{\mathsf{UPSI\text{-}del\text{-}alt}}$). We show that $\mathcal{F}_{\mathsf{UPSI\text{-}del\text{-}alt}}$ can be realized given $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$. That is, any protocol achieving $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$ can be easily transformed to achieve $\mathcal{F}_{\mathsf{UPSI\text{-}del\text{-}alt}}$. Intuitively, the idea is that given the output I_{d-1} of $\mathcal{F}_{\mathsf{UPSI\text{-}del\text{-}alt}}$ on Day $(d-1)$, to obtain the output on Day d, we essentially need to do two things: (i) Add to I_{d-1} the contribution of the new inputs X_d and Y_d, (ii) Remove from I_{d-1} the contribution of the deleted data X_{d-t} and Y_{d-t}. Observe that (i) is exactly the output of $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$ on Day d. For (ii), from the output I_{d-1} and its own inputs, P_0 can compute $A = X_{d-t} \cap (Y_{[d-1]} \setminus Y_{[d-t-1]})$ which is the contribution of X_{d-t} to I_{d-1}. Similarly, P_1 can compute Y_{d-t}'s contribution $B = Y_{d-t} \cap (X_{[d-1]} \setminus X_{[d-t-1]})$. Then, they can simply exchange this information with each other in plaintext and this completes (ii). This exchange doesn't leak extra any information because, from the output of the functionality, $I_{d-1} \setminus I_d$ is in fact $(A \dot{\cup} B)$. From this, and the knowledge of A that P_0 can compute locally, P_0

can automatically learn B in the ideal world. Similarly for P_1. For completeness, we describe the protocol formally in Figure 11.

Notation. For any $i \leq 0$, define $X_i = Y_i = \emptyset$. Let $I_0 = \emptyset$.
Day d: P_0 has new input X_d and P_1 has new input Y_d. P_0's and P_1's inputs over the last t days are $\big(X_{d-t+1}, \ldots, X_{d-1}, X_d\big)$ and $\big(Y_{d-t+1}, \ldots, Y_{d-1}, Y_d\big)$ respectively. The protocol works as follows:

1. Invoke $\mathcal{F}_{\mathsf{UPSI\text{-}del\text{-}alt}}$ with P_0's input on Day d as X_d and P_1's input as Y_d. Both parties learn $I_z = \Big(X_d \cap (Y_{[d]} \setminus Y_{[d-t]})\Big) \cup \Big(Y_d \cap (X_{[d]} \setminus X_{[d-t]})\Big)$.
2. Let I_{d-1} be the output on Day $(d-1)$. From I_{d-1}, P_0 deduces and sends $A = X_{d-t} \cap (Y_{[d-1]} \setminus Y_{[d-t-1]})$. Similarly, P_1 deduces and sends $B = Y_{d-t} \cap (X_{[d-1]} \setminus X_{[d-t-1]})$.
3. Both parties output $I_d = I_z \cup \big(I_{d-1} \setminus (A \cup B)\big)$.

Figure 11: Protocol satisfying $\mathcal{F}_{\mathsf{UPSI\text{-}del\text{-}alt}}$ in the $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$-hybrid model.

Motivating example for $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$. Finally, our motivating example for studying updatable PSI with weak deletion also holds for $\mathcal{F}_{\mathsf{UPSI\text{-}del}}$. In privacy-preserving contact tracing, consider the scenario where one party's (server's) input is the set of people who tested positive on that day, the other party's (client's) input is the set of people they interacted with on that day. The output on each day is the union of two parts: (a) people who tested positive in the last t days intersecting with those clients met on that day, and (b) people who tested positive on that day intersecting with those clients met in the last t days. Essentially, this captures whether client is at risk of having been infected.

7 Experimental Results

We implement our two-sided and one-sided UPSI with addition protocols ($\Pi_{\mathsf{UPSI\text{-}add\text{-}two}}$ and $\Pi_{\mathsf{UPSI\text{-}add\text{-}one}}$) in C++ and report their performance in this section.

7.1 Implementation Details

We set the computational security parameter to $\lambda = 128$ and statistical security parameter to $\sigma = 40$. We use the CryptoTools library [Rin] for our underlying cryptographic primitives. In particular, we use the Boost library [Boo] for networking, the Relic library [AGM+] for the instantiation of elliptic curves, and SHA256 from OpenSSL [Ope] for the hash functions.

We compare our UPSI with addition protocols with the state-of-the-art OT extension based semi-honest PSI protocols which are optimized for different network settings:

- KKRT16 [KKRT16]: computation-optimized and works best in the setting of LAN networks.
- SpOT-Light [PRTY19]: communication-optimized and works best in networks with low bandwidth. They have two variants of the protocol, a speed-optimized variant (spot-fast) and a communication-optimized variant (spot-low). We compare our protocols with both variants.

- CM20 [CM20]: balanced between computation and communication, and works best in networks with moderate bandwidth (e.g., $30 - 100$ Mbps).

We run all the experiments between two virtual machines with Intel(R) Xeon(R) 3.0 GHz CPU and 32 GB RAM, which communicate over a LAN network. We simulate the WAN connection using the Linux tc command, where the RTT latency is set to be 80 ms and we test on various network bandwidths. All of our experiments use a single thread for each party.

Setting. To demonstrate the updatable property, we consider the following setting: each party initially holds an empty set. Then, on every new Day d, both parties add a new set of size N_d to their existing sets and wish to learn the updated set intersection. We repeat this process over a period of several days ($\frac{N}{N_d}$) till the total set size of each party is N. We compare the *amortized* (over the total number of days) communication cost and running time of our protocol with the prior PSI protocols [KKRT16, PRTY19, CM20], where, on any Day d, the two parties run a fresh PSI on their updated sets to learn the updated intersection.

7.2 Two-Sided UPSI with Addition

We implement the two-sided UPSI with addition protocol $\Pi_{\mathsf{UPSI\text{-}add\text{-}two}}$ presented in Section 4, where the PSI protocol in Step 3b is instantiated with a DDH-based PSI [Mea86, HFH99]. A detailed comparison for $N = 2^{16} - 2^{22}$ and $N_d = 2^8 - 2^{12}$ is presented in Table 2. Note that for the PSI protocols [KKRT16, PRTY19, CM20], we only report for $N_d = 2^8$ because both their communication and running time are dominated by N (which is much larger than N_d) and do not differ much for other N_d values.

Communication Improvement. The communication cost of our protocol on any day is proportional only to the update size N_d and independent of the size of the entire set (that grows gradually to N), whereas all the PSI protocols require communication to grow with the entire set. Therefore, our protocol beats all the PSI protocols in amortized communication by $7.5 - 13250\times$ in the settings we consider (where $N \gg N_d$).

Computation Improvement. Similar to communication, our computational cost also grows only with N_d while all the PSI protocols require computation to grow with the size of the entire set (that gradually grows to N). However, our protocol does not beat their computation in all the settings because all these PSI protocols only use OT extension [IKNP03, ALSZ13] along with symmetric cryptographic primitives (AES/hash functions), which are computationally very efficient, while our protocol requires public-key operations. As a result, our protocol is computationally more expensive for smaller values of N but eventually beats all these protocols when N is sufficiently large. In particular, for $N = 2^{22}$ and $N_d = 2^8$, our protocol beats [KKRT16] (the computationally most efficient protocol) by $2.6\times$ in computation.

Overall Running Time. Generally speaking, our protocol has more advantages in the total running time when the network bandwidth is lower, the total set size N is larger, and the update size N_d is smaller. For example, if we focus on the setting $N = 2^{20}$, when $N_d = 2^8$, our protocol beats the best PSI protocol by $1.1 - 24.5\times$ for network bandwidth between $5 - 200$ Mbps; when $N_d = 2^{10}$, our protocol beats the best PSI protocol by $1.1 - 7.6\times$ for network bandwidth between

N	N_d	Protocol	Comm. (MB)	Total Running Time (s)			
				LAN	200Mbps	50Mbps	5Mbps
2^{16}	2^8	KKRT16	3.90	**0.05**	**1.01**	1.32	7.11
		spot-fast	2.32	0.91	1.38	1.38	7.06
		spot-low	1.96	4.99	5.23	5.39	6.93
		CM20	2.65	0.29	1.23	**1.30**	**4.94**
	2^8	Ours	0.02	1.65	2.55	2.64	**2.66**
	2^{10}		0.06	6.06	6.81	7.22	8.13
	2^{12}		0.26	23.5	24.2	25.7	29.5
2^{18}	2^8	KKRT16	15.9	**0.25**	1.92	3.43	27.5
		spot-fast	9.45	3.49	4.08	4.20	**12.8**
		spot-low	7.80	21.2	22.8	23.3	30.4
		CM20	10.7	0.90	**1.73**	**2.81**	18.6
	2^8	Ours	0.02	1.65	2.55	**2.63**	**2.66**
	2^{10}		0.06	6.07	6.84	7.26	**8.15**
	2^{12}		0.26	23.6	24.2	25.8	29.6
2^{20}	2^8	KKRT16	64.2	**1.03**	**2.89**	12.7	109
		spot-fast	38.2	12.2	13.1	13.7	65.1
		spot-low	31.6	110	107	113	146
		CM20	43.8	3.50	4.41	**8.43**	74.6
	2^8	Ours	0.02	1.65	**2.55**	**2.63**	**2.66**
	2^{10}		0.06	6.08	6.85	**7.29**	**8.57**
	2^{12}		0.26	23.6	24.3	25.7	**30.0**
2^{22}	2^8	KKRT16	265	**4.30**	**8.71**	49.1	441
		spot-fast	157	49.7	51.3	54.8	196
		spot-low	—	—	—	—	—
		CM20	178	15.0	16.2	31.6	303
	2^8	Ours	0.02	**1.65**	**2.55**	**2.63**	**2.66**
	2^{10}		0.06	6.08	**6.85**	**7.29**	**8.84**
	2^{12}		0.26	23.6	24.3	**25.8**	**30.2**

Table 2: Amortized communication cost (in MB) and running time (in seconds) comparing our protocol $\Pi_{\mathsf{UPSI\text{-}add\text{-}two}}$ to [KKRT16], spot-fast and spot-low [PRTY19], and [CM20]. The LAN network has 20 Gbps bandwidth and 0.1 ms RTT latency. All the other network settings have 80 ms RTT. Cells with "—" denote settings where the programs run out of memory and those in blue indicate the fastest running time for that setting.

$5 - 50$ Mbps; when $N_d = 2^{12}$, our protocol beats the best PSI protocol by $2.1\times$ for network bandwidth 5 Mbps. On the other hand, for the setting where $N = 2^{22}$, when $N_d = 2^8$, our protocol beats the best PSI protocol by $2.6 - 73.7\times$ for all networks.

7.3 One-Sided UPSI with Addition

We implement the one-sided UPSI with addition protocol $\Pi_{\mathsf{UPSI\text{-}add\text{-}one}}$ presented in Section 5 with the optimizations mentioned in Section 5.5. We pick the Cuckoo hashing parameters according to

Pinkas et al. [PSSZ15]. In Figure 6, we set the batch size for both parties (the number of elements added each day) to be $2^6 = 64$ instead of σ (which is 40).[4] To insert $n = 4 \cdot 2^6 = 2^8$ elements into the Cuckoo hash table, we set the number of bins as $1.2n = 308$ and stash size as 12. A detailed comparison for $N = 2^{16} - 2^{20}$ and $N_d = 2^6 - 2^{10}$ is presented in Table 3.[5] For the PSI protocols [KKRT16,PRTY19,CM20], we only report for $N_d = 2^8$ as their amortized communication and running time are dominated by N and do not differ much for other N_d values.

N	N_d	Protocol	Comm. (MB)	Total Running Time (s)			
				LAN	200Mbps	50Mbps	5Mbps
2^{16}	2^8	KKRT16	3.90	**0.05**	**1.01**	1.32	7.11
		spot-fast	2.32	0.91	1.38	1.38	7.06
		spot-low	1.96	4.99	5.23	5.39	6.93
		CM20	2.65	0.29	1.23	**1.30**	**4.94**
	2^6	Ours	0.30	2.96	3.46	3.55	**3.62**
	2^8		0.97	10.6	11.9	12.0	12.1
	2^{10}		2.95	35.5	37.6	37.7	37.8
2^{18}	2^8	KKRT16	15.9	**0.25**	1.92	3.43	27.5
		spot-fast	9.45	3.49	4.08	4.20	**12.8**
		spot-low	7.80	21.2	22.8	23.3	30.4
		CM20	10.7	0.90	**1.73**	**2.81**	18.6
	2^6	Ours	0.37	3.38	3.98	4.07	**4.16**
	2^8		1.21	12.5	14.1	14.2	14.3
	2^{10}		3.88	42.2	44.7	44.8	44.9
2^{20}	2^8	KKRT16	64.2	**1.03**	**2.89**	12.7	109
		spot-fast	38.2	12.2	13.1	13.7	65.1
		spot-low	31.6	110	107	113	146
		CM20	43.8	3.50	4.41	**8.43**	74.6
	2^6	Ours	0.43	3.88	4.58	**4.68**	**4.78**
	2^8		1.45	14.8	16.6	16.8	**16.9**
	2^{10}		4.84	50.6	53.6	53.7	**53.8**

Table 3: Amortized communication cost (in MB) and running time (in seconds) comparing our protocol $\Pi_{\mathsf{UPSI\text{-}add\text{-}one}}$ to [KKRT16], spot-fast and spot-low [PRTY19], and [CM20]. The LAN network has 20 Gbps bandwidth and 0.1 ms RTT latency. All the other network settings have 80 ms RTT. Cells with "—" denote settings where the programs run out of memory and those in blue indicate the fastest running time for that setting.

Communication Improvement. The amortized communication cost of our protocol grows linearly only with the update size N_d and logarithmically with the size of the entire set (that grows gradually to N), whereas all the PSI protocols require communication to grow linearly with the entire set. Therefore, our protocol beats these PSI protocols in amortized communication by $2-149\times$

[4] We use 2^6 instead of 40 for two reasons: In the parameters from [PSSZ15], the stash size is available only for $n = 2^8$ and not lower numbers. Also, since we consider daily updates that are powers of 2, running batches of 2^6 is more convenient than 40.

[5] Unlike Table 2, we don't include $N = 2^{22}$ as we ran out of memory for that case.

in almost all the settings we consider, the only exception being when $N = 2^{16}$ and $N_d = 2^{10}$.

Computation Improvement. Our amortized computational cost also grows only linearly with N_d and logarithmically with the size of the entire set (that grows gradually to N) while all the PSI protocols require computation to grow linearly with the entire set. However, our protocol does not beat [KKRT16] (the computationally most efficient protocol) or [CM20] in the settings we consider because N is not sufficiently large. In particular, we expect our protocol to beat both [KKRT16, CM20] in computation when $N = 2^{22}$ and $N_d = 2^6$ (we currently run out of memory for $N = 2^{22}$). We also note that for $N = 2^{20}$ and $N_d = 2^6$, our protocol beats [PRTY19] by $3.1 - 28.3\times$ in computation.

Overall Running Time. Generally, our protocol has more advantages in the total running time when the network bandwidth is lower, the total set size N is larger, and the update size N_d is smaller. For example, if we focus on the setting $N = 2^{20}$, when $N_d = 2^6$, our protocol beats the best PSI protocol by $1.8 - 30.5\times$ for network bandwidth between $5 - 50$ Mbps; when $N_d = 2^8$, our protocol beats the best PSI protocol by $3.9\times$ for network bandwidth 5 Mbps; when $N_d = 2^{10}$, our protocol beats the best PSI protocol by $1.2\times$ for network bandwidth 5 Mbps.

References

[AGM+] D. F. Aranha, C. P. L. Gouva, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS*, 2013.

[APP] Password Monitoring – Apple Platform Security. https://support.apple.com/en-al/guide/security/sec78e79fc3b/web.

[ATD20] Aydin Abadi, Sotirios Terzis, and Changyu Dong. Feather: Lightweight multi-party updatable delegated private set intersection. *IACR Cryptol. ePrint Arch.*, 2020:407, 2020.

[BKM+20] Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. Private matching for compute. *IACR Cryptol. ePrint Arch.*, 2020.

[Boo] Boost C++ Libraries. https://www.boost.org/.

[BPSW07] Justin Brickell, Donald E Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *CCS*, 2007.

[CCF+20] Justin Chan, Landon P. Cox, Dean P. Foster, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham M. Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Puneet Sharma, Sudheesh Singanamalla, Jacob E. Sunshine, and Stefano Tessaro. PACT: privacy-sensitive protocols and mechanisms for mobile contact tracing. *IEEE Data Eng. Bull.*, 2020.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *CCS*, 2017.

[CM20] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In *CRYPTO*, 2020.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, 1984.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[HFH99] Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *ACM Conference on Electronic Commerce (EC-99)*, 1999.

[IKN+20] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *EuroS&P*, 2020.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, 2003.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *CCS*, 2016.

[KLS+17] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. *Proc. Priv. Enhancing Technol.*, 2017(4):177–197, 2017.

[KRS+19] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *USENIX Security*, 2019.

[Mea86] C. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *IEEE S & P*, 1986.

[MIC] Password Monitor: Safeguarding passwords in Microsoft Edge. https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/.

[Ope] OpenSSL. https://www.openssl.org/.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 2004.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In *CRYPTO*, 2019.

[PRTY20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from paxos: Fast, malicious private set intersection. In *EUROCRYPT*, 2020.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX*, 2015.

[PSTY19] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *EUROCRYPT*, 2019.

[PSWW18] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In *EUROCRYPT*, 2018.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on ot extension. In *USENIX*, 2014.

[PSZ18] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018.

[Rin] Peter Rindal. CryptoTools: a portable library containing a collection of tools for building cryptographic protocols. `https://github.com/ladnir/cryptoTools`.

[RR17] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In *CCS*, 2017.

[SvDS+18] Emil Stefanov, Marten van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. *J. ACM*, 2018.

[TPKC07] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Celik. Privacy preserving error resilient dna searching through oblivious automata. In *CCS*, 2007.

[TPY+19] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. Protecting accounts from credential stuffing with password breach alerting. In *USENIX Security*, 2019.

[TSS+20] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight contact tracing with strong privacy. *IEEE Data Eng. Bull.*, 2020.