Updatable Private Set Intersection Revisited: Extended Functionalities, Deletion, and Worst-Case Complexity

Saikrishna Badrinarayanan[1], Peihan Miao[2], Xinyi Shi[2], Max Tromanhauser[2], and Ruida Zeng[2]

[1]LinkedIn
[2]Brown University

Abstract

Private set intersection (PSI) allows two mutually distrusting parties each holding a private set of elements, to learn the intersection of their sets without revealing anything beyond the intersection. Recent work (Badrinarayanan et al., PoPETS'22) initiates the study of updatable PSI (UPSI), which allows the two parties to compute PSI on a regular basis with sets that constantly get updated, where both the computation and communication complexity only grow with the size of the small updates and not the large entire sets. However, there are several limitations of their presented protocols. First, they can only be used to compute the plain PSI functionality and do not support extended functionalities such as PSI-Cardinality and PSI-Sum. Second, they only allow parties to add new elements to their existing set and do not support arbitrary deletion of elements. Finally, their addition-only protocols either require both parties to learn the output or only achieve low complexity in an amortized sense and incur linear worst-case complexity.

In this work, we address all the above limitations. In particular, we study UPSI with semi-honest security in both the addition-only and addition-deletion settings. We present new protocols for both settings that support plain PSI as well as extended functionalities including PSI-Cardinality and PSI-Sum, achieving one-sided output (which implies two-sided output). In the addition-only setting, we also present a protocol for a more general functionality Circuit-PSI that outputs secret shares of the intersection. All of our protocols have worst-case computation and communication complexity that only grow with the set updates instead of the entire sets (except for a polylogarithmic factor). We implement our new UPSI protocols and compare with the state-of-the-art protocols for PSI and extended functionalities. Our protocols compare favorably when the total set sizes are sufficiently large, the new updates are sufficiently small, or in networks with low bandwidth.

Keywords: Private Set Intersection, Secure Two-Party Computation, Oblivious Data Structure.

1 Introduction

Private Set Intersection (PSI) enables two distrusting parties, each holding a private set of elements, to jointly compute the intersection of their sets without revealing anything other than the intersection itself. Despite its simple functionality, PSI and its related notions have found many real-world applications including online advertising measurement (deployed by Google Ads [IKN+20, Ads]), secure password breach alert (deployed by Google Chrome [Chr], Microsoft Edge [MIC], Apple iCloud Keychain [Key], etc.), mobile private contact discovery (deployed by Signal [KRS+19, Sig]), privacy-preserving contact tracing in a global pandemic (jointly deployed by Google and Apple [TSS+20, BBV+20, App]). The last several decades have witnessed enormous progress towards realizing PSI efficiently using various techniques achieving both semi-honest and malicious security [KKRT16, CLR17, PSTY19, PRTY20, CMdG+21, GPR+21, CRR21, CGS22, RR22, CILO22, BPSY23].

In many real-world applications such as aggregated ads measurement and privacy-preserving contact tracing, PSI is performed on a regular (e.g., daily) basis with *updated* sets, where the updates can be small when compared to the entire sets. However, most of the existing work requires the two parties to perform a fresh PSI protocol every time. A recent work by Badrinarayanan et al. [BMX22] initiates the study of *updatable PSI (UPSI)*, which allows the two parties to compute set intersections for sets that regularly get updated. Their work presents protocols for updatable PSI where both the computation and communication complexity only grow with the size of the updates and are independent of the size of the entire sets (except for a logarithmic factor). As a result, these protocols are orders of magnitudes faster than a fresh PSI protocol, especially when the updates are significantly smaller than the entire sets. Nevertheless, there are several limitations with the protocols in [BMX22].

- **Functionality:** All the protocols presented in [BMX22] are restricted to the *plain* PSI functionality, crucially leveraging the fact that parties learn all the elements in the intersection. However, certain real-world applications require more refined PSI functionalities that do not reveal the entire intersection but instead only provide aggregated information about the intersection or enable restricted computation on the data in the intersection. As two specific examples that model many applications such as online advertising measurement, *PSI-Cardinality* allows two parties to jointly learn the cardinality (or size) of their set intersection; *PSI-Sum* allows two parties, where one party additionally holds a private integer value associated with each element in her set, to jointly compute the sum of the associated integer values for all the elements in the intersection (together with the cardinality of the intersection).

- **Addition-Only:** [BMX22] mainly focuses on the addition-only setting, where both parties can only *add* new elements to their existing old sets, and do not support arbitrary *deletion* of elements from their sets. Note that they present a protocol for *UPSI with weak deletion*, which allows the 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. However, it does not support arbitrary deletion, and the daily computation and communication complexity additionally grows with t.

- **Tradeoffs of the Addition-Only Protocols:** [BMX22] presents two protocols for addition-only UPSI, each with its own tradeoffs. In particular, one protocol crucially requires *both* parties to learn the output (namely, two-sided UPSI), which may not be applicable in certain applications such as password breach alert. The other protocol allows a single party to learn the output (namely, one-sided UPSI), but it only achieves low computation and communication complexity in an *amortized* sense over many days; the *worst-case* complexity can be as high as linear in the entire sets. Note that one-sided UPSI is a strictly stronger functionality in the semi-honest setting (as considered in [BMX22]) since the output-receiving party can simply send the output to the other party so as to achieve two-sided UPSI.

1.1 Our Results

In this work, we address all the aforementioned limitations by presenting new UPSI protocols for extended functionalities, supporting both addition and deletion of elements, achieving one-sided output and low worst-case complexity in both computation and communication. All of our protocols are secure in the semi-honest model, hence one-sided UPSI is a stronger functionality. In the setting with both addition and deletion, we achieve a slightly more general functionality than PSI-Sum as defined in [IKN+20, MPR+20], where we do *not* reveal the cardinality of the intersection along with the sum.

Besides the functionalities of plain PSI, PSI-Cardinality, and PSI-Sum that we discussed above, we consider a more general functionality of Circuit-PSI [PRTY19, RS21, CGS22, RR22, BPSY23], where the two parties learn the cardinality of the intersection as well as an additive secret share of each element in it. This functionality allows the two parties to perform further computation over the shares afterwards.

Note that we only consider Circuit-PSI in the addition-only setting. The challenge in achieving Circuit-PSI with both addition and deletion is as follows. Intuitively speaking, when deleting elements from the intersection, the parties must learn which existing secret shares to delete from the intersection (unless the parties update their entire secret shared intersection, where the complexity grows with the entire sets, which is undesirable). Given that they know *when* a particular secret share (not the element itself) was added to the intersection, this essentially reveals more information than what the ideal functionality outputs. Crucially, note that in the case of plain PSI with addition and deletion, this is not a problem since the ideal functionality's output also reveals *when* a particular element was added and deleted; and in the case of PSI-Cardinality or PSI-Sum, parties only learn aggregated information and this challenge doesn't arise in the protocol design. We summarize our results in comparison with [BMX22] in Table 1.

<table>
<tr><th>Protocol</th><th>Functionality</th><th>Output</th><th>Addition/Deletion</th><th colspan="2">Comp. & Comm. Complexity</th></tr>
<tr><td>[BMX22, $\Pi_{\textsf{UPSI-add-two}}$]</td><td>PSI</td><td>Two-Sided</td><td>Addition-Only</td><td colspan="2">$O(N_d)$</td></tr>
<tr><td>[BMX22, $\Pi_{\textsf{UPSI-add-one}}$]</td><td>PSI</td><td>One-Sided</td><td>Addition-Only</td><td colspan="2">$O^*(N_d \cdot \log N)$</td></tr>
<tr><td>Figure 11, $\Pi_{\textsf{UPSI-Add}_{\textsf{psi}}}$</td><td>PSI</td><td rowspan="3">One-Sided</td><td rowspan="4">Addition-Only</td><td colspan="2" rowspan="4">$O(N_d \cdot \log N)$</td></tr>
<tr><td>Figure 5, $\Pi_{\textsf{UPSI-Add}_{\textsf{ca}}}$</td><td>PSI-Cardinality</td></tr>
<tr><td>Figure 5, $\Pi_{\textsf{UPSI-Add}_{\textsf{sum}}}$</td><td>PSI-Sum</td></tr>
<tr><td>Figure 5, $\Pi_{\textsf{UPSI-Add}_{\textsf{circuit}}}$</td><td>Circuit-PSI</td><td>Secret Shared</td></tr>
<tr><td>[BMX22, $\Pi_{\textsf{UPSI-del}}$]</td><td>PSI</td><td>Two-Sided</td><td>Weak Deletion</td><td colspan="2">$O(N_d \cdot t)$</td></tr>
<tr><td>Figure 10, $\Pi_{\textsf{UPSI-Del}_{\textsf{psi}}}$</td><td>PSI</td><td rowspan="3">One-Sided</td><td rowspan="3">Addition & Deletion</td><td rowspan="3">Single Deletion
$O(N_d \cdot \log N)$</td><td rowspan="3">Arbitrary Deletion
$O(N_d \cdot \log^2 N)$</td></tr>
<tr><td>Figure 10, $\Pi_{\textsf{UPSI-Del}_{\textsf{ca}}}$</td><td>PSI-Cardinality</td></tr>
<tr><td>Figure 10, $\Pi_{\textsf{UPSI-Del}_{\textsf{sum}}}$</td><td>PSI-Sum†</td></tr>
</table>

Table 1: Summary of our results in comparison to [BMX22], including functionality, one-sided or two-sided output, support of addition and deletion of elements, and computation and communication complexity. PSI-Sum† denotes the variant of PSI-Sum that does not reveal the cardinality. N denotes the size of the entire sets and N_d denotes the size of the d-th update. t denotes the number of updates when parties refresh their sets in UPSI with weak deletion. $O^*(\cdot)$ denotes amortized complexity. For UPSI with both addition and deletion, we present two variants, one allowing each element to be added and deleted at most once, and the other allowing arbitrary additions and deletions of the same element.

Experiments. We implement all our protocols and compare their performance with the state-of-the-art protocols for PSI and extended functionalites [CGS22, RR22]. As our communication grows with the size of the update and not the entire input (except by a logarithmic factor), we demonstrate a significant improvement, up to orders of magnitude, when the input sets grow sufficiently large with smaller updates. Although our usage of public key operations dampens the asymptotic impact on computation, in realistic WAN settings, our protocols are able to outperform prior work in end-to-end running time. We also compare our new one-sided addition-only UPSI protocol with [BMX22] and show significant improvement in worst-case complexity.

1.2 Technical Overview

We discuss the technical challenges and novelties in this work. We start with addition-only UPSI. Let X, Y denote the old sets of the two parties P_0, P_1 respectively, and let X_d, Y_d denote their new added sets on Day d. For simplicity, assume $|X| = |Y| = N$ and $|X_d| = |Y_d| = N_d$.[1] Recall that we are mostly interested in the scenario when the set updates are significantly smaller than the entire sets, namely $N \gg N_d$. The parties have already learned $I = X \cap Y$ of the old sets, and they would like to learn the updated intersection $I_d = (X \cup X_d) \cap (Y \cup Y_d)$. We focus on one-sided UPSI, where only P_0 learns the output.

Addition-Only UPSI with Extended Functionalities. Our starting point is the one-sided addition-only UPSI protocol in [BMX22]. They observe that it suffices to learn the set difference $I_d \setminus I$ on each day, which, from P_0's perspective, can be split into two disjoint sets, $(X_d \cap (Y \cup Y_d))$ and $(X \cap Y_d)$. They then develop protocols to compute the two sets individually, with complexity growing only with N_d and not N. To compute UPSI-Cardinality, we similarly split $|I_d \setminus I|$ into $|X_d \cap (Y \cup Y_d)|$ and $|X \cap Y_d|$, and compute them individually. Note that this is not sufficient since the individual cardinalities reveal more information than the ideal functionality, which we will fix later.

Computing $|X_d \cap (Y \cup Y_d)|$: We first briefly describe the approach in [BMX22] to computing $X_d \cap (Y \cup Y_d)$. Their key idea is to let P_1 store an encrypted version of her set on P_0's side; on each day, she updates this encrypted dataset based only on her new input Y_d. Here, they require a data structure that allows P_1 to obliviously update the dataset and P_0 to obliviously query and compute on the dataset. [BMX22] constructs such an oblivious data structure via a binary tree and uses additively homomorphic encryption to compute on encrypted data. By carefully re-crafting the homomorphic operations on the encrypted data in the oblivious data structure, we design a method that reveals only the number of elements that are matched between X_d and the encrypted dataset $(Y \cup Y_d)$. This enables P_0 to learn $|X_d \cap (Y \cup Y_d)|$.

Computing $|X \cap Y_d|$: We review the approach in [BMX22] to computing $X \cap Y_d$, which leverages Diffie-Hellman-based PSI in [BMX22]. Unfortunately, it does not extend to updatable cardinality. To address this challenge, our idea is to compute $|X \cap Y_d|$ symmetrically on P_1's side using the oblivious data structure. In particular, we let P_0 store an encrypted version of his set on P_1's side that supports efficient and oblivious updates and queries. This way we can efficiently allow P_1 to learn $|X \cap Y_d|$.

Computing the sum with one-sided output: There are two issues with our current approach: first, individual cardinalities should *not* be revealed to the parties; second, P_1 should *not* learn anything about the output. At a high level, P_0 learns the cardinality $|X_d \cap (Y \cup Y_d)|$ by decrypting a set of (homomorphically evaluated) ciphertexts and counts the number of 0's in them. This

[1] Our constructions work for two sets with different sizes as well, which we elaborate in Section 3 and Section 4.

happens similarly for P_1 to learn $|X \cap Y_d|$. To fix the first issue, we develop a method to combine the two sets of ciphertexts, re-randomize and shuffle all of them, and then decrypt them at the end. The number of 0's reveals only the sum of $|X_d \cap (Y \cup Y_d)|$ and $|X \cap Y_d|$, rather than individual values. To fix the second issue, we use a 2-out-of-2 threshold encryption scheme. The parties will jointly decrypt all the ciphertexts only after the random shuffling, and the decrypted results are revealed only to P_0. This protocol can be further extended to PSI-Sum and Circuit-PSI by attaching a payload to each element and further leveraging additive homomorphism.

Worst-Case Logarithmic Complexity. The above construction relies heavily on the oblivious data structure presented in [BMX22]. A critical drawback of the data structure is that it only achieves logarithmic complexity in an *amortized* sense, namely the average complexity over many days is low. However, the *worst-case* complexity can be as high as linear in the entire sets. In this work, we construct a new oblivious data structure with worst-case logarithmic complexity.

Recall that in our UPSI construction, P_1 store an encrypted version of her set, maintained in an oblivious data structure, on P_0's side. There are two requirements on the data structure: first, for each new element y added to P_1's set, P_1 can update the encrypted dataset without leaking any information about y to P_0; second, for each new element x added to P_0's set, P_0 can locally identify a small set of encryptions in the P_1's set that are potential matches to x.

At a high level, our construction works as follows. The encrypted dataset is maintained in a binary tree structure. Each element x identifies a designated, (pseudo)random root-to-leaf path, computed by a pseudorandom function $F_{\mathsf{k}}(x)$ with k known to both parties. As P_1 updates the tree, she will maintain the invariant that each element y always appears along its designated path. This allows P_0 to query for potential matches by collecting all elements in the appropriate path (i.e., potential matches to x will be found in the path designated by $F_{\mathsf{k}}(x)$). However, when a new element y is added to P_1's set, directly updating the designated path of y in P_0's storage reveals information about y being added to the tree. *Therefore, we need a mechanism for P_1 to add y to its designated path in P_0's storage while hiding the path from P_0.* In [BMX22], this is achieved through a series of operations that update an entire level of the tree each time, resulting in an amortized logarithmic complexity, while the worst-case complexity is linear (when P_1 updates the leaf level of the tree).

Our solution takes inspiration from the Path ORAM construction [SvS+13]. Instead of updating the designated path, P_1 picks a random path each time, and "pushes down" the elements along that path as much as possible. The access pattern of tree updates consist of random paths, hence are oblivious to P_0. Note that Path ORAM has an additional logarithmic factor from tree recursions due to limited registers. We can remove the tree recursions since we do not have this restriction in UPSI, leading to a single logarithmic factor. We refer to Section 3 for more details of our addition-only UPSI protocols.

Supporting Deletion. Our oblivious data structure is inspired by ORAM, but the manner in which ORAM handles deletion (or modification) of memory content does not work for us. In Path ORAM, whenever x is accessed (or modified), x will be re-allocated to a new, freshly sampled random designated path. However, as discussed above, the designated path of x in our construction is fixed and known to both parties.

Our key idea is to keep the fixed designated path for the element and attach a payload of $+1$ or -1 to indicate addition or deletion. Specifically, when y is deleted from P_1's set, instead of deleting it from the data structure, she will *add* another y to the data structure with a payload of -1 indicating deletion. In other words, when y is added *or* deleted from P_1's set, she will add a new pair of encryptions $(\mathsf{Enc}(y), \mathsf{Enc}(+1))$ or $(\mathsf{Enc}(y), \mathsf{Enc}(-1))$ to the designated path of y. Recall that we can update the tree by accessing a random path, hence the access pattern remains oblivious to P_0. When x is added to P_0's set, P_0 will still identify all the encrypted pairs on the designated path of x as potential matches. However, the crucial challenge is when y is not in the intersection, we need to further hide from P_0 whether y was never added to the dataset, or y was added and then deleted (namely, $(y, +1)$ and $(y, -1)$ cancel out). To achieve this, we design a special protocol that, for each pair, if the element is a match, then the parties obtain a secret share of its corresponding payload ($+1$ or -1); otherwise they obtain a secret share of 0. Finally, they add up all these secret shares where $+1$'s and -1's are canceled out, revealing whether x is in the intersection.

There are several other challenges that arise in handling deletions. For instance, we need to bound the maximum node size of the tree, especially when there are unlimited, repeated elements being added to the same path. If we restrict each element to being added and deleted at most once, the complexity remains the same as in the addition-only protocols. A more nuanced analysis shows that with unlimited additions and deletions, the complexity incurs only an additional logarithmic factor. Another challenge arises in plain UPSI, when P_0 removes x and P_1 adds $y = x$ on the same day. After these updates, x is not in the intersection, and it should be further hidden that it was added and then deleted from the intersection. We refer to Section 4 for more details of how to handle these challenges and the full description of our UPSI protocols with both addition and deletion.

1.3 Related Work

There has been a long line of work towards realizing PSI efficiently using various techniques including Diffie-Hellman-based [Mea86, HFH99, IKN+20], RSA-based [DT10, ADT11], circuit-based [HEK12, PSSZ15, PSWW18, PSTY19], oblivious transfer (OT)-based [DCW13, PSZ14, KKRT16, PRTY19, CM20], fully homomorphic encryption (FHE)-based [CLR17, CHLR18, CMdG+21], and vector oblivious linear evaluation (VOLE)-based [RS21, GPR+21, CRR21, RR22, BPSY23] approaches, achieving both semi-honest and malicious security [RR17, OOS17, CHLR18, PRTY20, CILO22, RR22, BPSY23].

As discussed earlier, certain applications require PSI with extended functionalities that do not reveal the entire intersection but rather enable restricted computation on the elements in the intersection. PSI-Cardinality and PSI-Sum model many applications such as aggregated ads measurement [IKN+20, MPR+20] and privacy-preserving contact tracing [TSS+20, BBV+20]. More generally, Circuit PSI [HEK12, PSTY19, RS21, CGS22, RR22, BPSY23] enables the two parties to learn secret shares of the set intersection, which can be used to securely compute any function using generic secure two-party computation protocols [Yao86, GMW87]. However, all these approaches study PSI or PSI with extended functionalities in the standalone setting, which do not support small updates to the sets beyond running a fresh protocol after each update.

To the best of our knowledge, [BMX22] is the first work that formalizes and studies PSI in the updatable setting, which we have extensively discussed above. Another related work is [ADMT22], which studies delegatable PSI with small updates. Specifically, they allow multiple clients to outsource their (encrypted) private sets and delegate PSI computation to a cloud server. Clients can perform efficient updates on their outsourced sets where the computation and communication only grow with their updates. However, both the computation and communication costs of computing

PSI still grow with size of the entire sets, and their protocol crucially requires the existence of a server.

Concurrent and Independent Work. A concurrent and independent work by Agarwal et al. [ACG+24] constructs a semi-honest secure UPSI protocol that supports arbitrary addition and deletion of elements. Their construction, which builds UPSI from a new variant of structured encryption (StE), achieves worst-case communication and computation complexity that grows linearly with the size of the updates and poly-logarithmically with the size of the entire sets. Their framework supports the plain PSI functionality with two-sided output, and focuses on feasibility. In contrast, our work additionally achieves the extended functionalities with one-sided output (which implies two-sided output), and demonstrates concrete efficiency.

2 Preliminaries

Notation. We use λ, κ to denote the computational and statistical security parameters, respectively. For an integer $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$. A 2-out-of-2 additive secret share of a value $x \in \mathbb{Z}_n$ is denoted as $(\llbracket x \rrbracket_0, \llbracket x \rrbracket_1)$ where $\llbracket x \rrbracket_0 \xleftarrow{\$} \mathbb{Z}_n$ and $\llbracket x \rrbracket_0 + \llbracket x \rrbracket_1 = x \mod n$. PPT stands for probabilistic polynomial time. By $\overset{c}{\approx}$ we mean two distributions are computationally indistinguishable.

Additively Homomorphic Encryption. An additively homomorphic encryption scheme is a public-key encryption sccheme that consists of a tuple of PPT algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ over message space $\mathcal{M}$ with correctness, chosen-plaintext attack (CPA) security, and linear homomorphism.

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$: On input of the security parameter, output a public key pk and a secret key sk.
- $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$: On input of a public key pk and a message $m \in \mathcal{M}$, output a ciphertext c.
- $m/\bot \leftarrow \mathsf{Dec}_{\mathsf{sk}}(c)$: On input of a secret key sk and a ciphertext c, output a plaintext m or the symbol $\bot$.
- $\mathsf{Enc}_{\mathsf{pk}}(m_0 + m_1) \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_0) \oplus \mathsf{Enc}_{\mathsf{pk}}(m_1)$: On input two ciphertexts of m_0, m_1 encrypted under pk, output a ciphertext for their sum.
- $\mathsf{Enc}_{\mathsf{pk}}(m_0 \cdot m_1) \leftarrow m_0 \odot \mathsf{Enc}_{\mathsf{pk}}(m_1)$: On input a plaintext message m_0 and a ciphertext of m_1 encrypted under pk, output a ciphertext for their product.

Threshold Additively Homomorphic Encryption. A $(2,2)$-threshold additively homomorphic encryption scheme consists of a tuple of PPT algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{PartDec}, \mathsf{FullDec})$ over message space $\mathcal{M}$.

- $(\mathsf{pk}, \mathsf{sk}_0, \mathsf{sk}_1) \leftarrow \mathsf{KeyGen}(1^\lambda)$: On input of the security parameter, output a public key pk and a pair of secret key shares sk_0 and sk_1.
- $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$: On input of a public key pk and a message $m \in \mathcal{M}$, output a ciphertext c.
- $\hat{c} \leftarrow \mathsf{PartDec}_{\mathsf{sk}_b}(c)$: On input a secret key share sk_b (for $b \in \{0,1\}$) and a ciphertext c, output a partially decrypted ciphertext $\hat{c}$.

- $m/\bot \leftarrow \mathsf{FullDec}_{\mathsf{sk}_b}(\hat{c})$: On input a secret key share sk_b (for $b \in \{0,1\}$) and a partially decrypted ciphertext $\hat{c}$ by the other secret key sk_{1-b}, output a plaintext m or the symbol $\bot$.

The scheme satisfies correctness and CPA security even given a secret key share sk_b for $b \in \{0,1\}$. It also supports linear homomorphic operations $\oplus$ and $\odot$.

Re-randomization. A re-randomization algorithm $\tilde{c} \leftarrow \mathsf{ReRand}_{\mathsf{pk}}(c)$ homomorphically adds an independently generated encryption of zero to c, resulting in a ciphertext $\tilde{c}$ that is indistinguishable from a fresh ciphertext encrypting the same message as c. We implicitly assume that each homomorphic operation is followed by a re-randomization process. This is required in our protocols to ensure that the randomness of the final ciphertext is independent of the randomness used in the original ciphertexts. For the popular (threshold) additively homomorphic encryption schemes such as exponential El Gamal encryption [ElG85] and Paillier encryption [Pai99], a homomorphically evaluated ciphertext can be made statistically identical to a fresh ciphertext. We refer to [ElG85, Pai99] for formal definitions of correctness and CPA security.

3 Addition-Only UPSI

3.1 Definition

In this section, we formalize the ideal functionality and security definition for addition-only UPSI. Consider two parties P_0 and P_1 who wish to run PSI on a daily basis with updated sets. In the addition-only setting, they each hold a private set and add new elements to their respective sets each day. They want to jointly compute their set intersection (or extended functionalities) on their updated sets without revealing anything beyond that. We formalize addition-only UPSI as a special case of secure two-party computation with a reactive functionality defined in Figure 1.

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

Day d**:**

- **Public Parameters**: The number of additions that P_0 and P_1 are performing: $|X_d|$ and $|Y_d|$, respectively.
- **Inputs:**
  P_0 inputs a set $X_d \subseteq \{0,1\}^*$ where $X_d \cap X = \emptyset$. In $\mathcal{F}_{\mathsf{UPSI\text{-}Add_{sum}}}$, X_d includes an integer value associated with each set member (i.e., v_i is associated with $x_i \in X_d$).
  P_1 inputs a set $Y_d \subseteq \{0,1\}^*$ where $Y_d \cap Y = \emptyset$.
- **Update**: On receiving the inputs from both parties, the ideal functionality updates $X = X \cup X_d$ and $Y = Y \cup Y_d$.
- **Output:**
  In $\mathcal{F}_{\mathsf{UPSI\text{-}Add_{psi}}}$, P_0 learns the intersection $I_d = X \cap Y$.
  In $\mathcal{F}_{\mathsf{UPSI\text{-}Add_{ca}}}$, P_0 learns the cardinality of the intersection $C_d = |X \cap Y|$.
  In $\mathcal{F}_{\mathsf{UPSI\text{-}Add_{sum}}}$, P_0 learns $C_d = |X \cap Y|$ and $V_d = \sum_{i:x_i \in X \cap Y} v_i$.
  In $\mathcal{F}_{\mathsf{UPSI\text{-}Add_{circuit}}}$, both parties learn $C_d = |X \cap Y|$. For each new element z being added to the intersection, P_0 learns $[\![z]\!]_0$ and P_1 learns $[\![z]\!]_1$ as an additive secret share for z.

Figure 1: Ideal functionalities for one-sided addition-only UPSI: $\mathcal{F}_{\mathsf{UPSI\text{-}Add_{psi}}}, \mathcal{F}_{\mathsf{UPSI\text{-}Add_{ca}}}, \mathcal{F}_{\mathsf{UPSI\text{-}Add_{sum}}}, \mathcal{F}_{\mathsf{UPSI\text{-}Add_{circuit}}}$.

Let $X_{[D]} = \{X_1, \ldots, X_D\}$ and $Y_{[D]} = \{Y_1, \ldots, Y_D\}$ be the inputs for P_0 and P_1 after D days, respectively. 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 (for $b \in \{0,1\}$) in the protocol Π at the end of D days, respectively. For a functionality $\mathcal{F}$, let $\mathcal{F}_b$ be the output for P_b in the D days. Note that $\mathcal{F}_1 = \perp$ in all the functionalities except for $\mathcal{F}_{\mathsf{UPSI\text{-}Add}_{\mathsf{circuit}}}$.

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

$$\left(\mathsf{View}_0^{\Pi,D}(X_{[D]}, Y_{[D]}), \mathsf{Out}_1^{\Pi,D}(X_{[D]}, Y_{[D]})\right) \overset{c}{\approx} \left(\mathsf{Sim}_0(1^\lambda, X_{[D]}, \mathcal{F}_0(X_{[D]}, Y_{[D]})), \mathcal{F}_1(X_{[D]}, Y_{[D]})\right),$$
$$\left(\mathsf{View}_1^{\Pi,D}(X_{[D]}, Y_{[D]}), \mathsf{Out}_0^{\Pi,D}(X_{[D]}, Y_{[D]})\right) \overset{c}{\approx} \left(\mathsf{Sim}_1(0^\lambda, Y_{[D]}, \mathcal{F}_1(X_{[D]}, Y_{[D]})), \mathcal{F}_0(X_{[D]}, Y_{[D]})\right).$$

Notation. Let $\Pi_{\mathsf{AHE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{PartDec}, \mathsf{FullDec})$ be a $(2,2)$-threshold additively homomorphic encryption scheme (see definition in Section 2) over plaintext space $\mathbb{Z}_q$ for a prime q. Without loss of generality we assume all the set elements are in $\mathbb{Z}_q$ (if not, we can apply a collision-resistant hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ on all the elements and perform PSI on the hash outputs). Let $F : \{0,1\}^\lambda \times \mathbb{Z}_q \to \{0,1\}^\lambda$ be a pseudorandom function (PRF). For a bit string $s \in \{0,1\}^n$, let $s_{[1:i]}$ denote the prefix of s of length i (for $i \in [n]$).

Consider a binary tree data structure with tree height L and 2^L leaves, let $\ell \in \{0, 1, \ldots, 2^L - 1\}$ denote the ℓ-th leaf node of the tree. Any leaf node ℓ defines a unique path from the root to the leaf. We use $\mathcal{P}(\ell)$ to denote such a path, and $\mathcal{P}(\ell, k)$ to denote the node in $\mathcal{P}(\ell)$ at level k of the tree (for $k \in \{0, 1, \ldots, L\}$). Let σ denote the maximum tree node size and ρ denote the stash size of our oblivious data structure.

3.2 Construction

In this section, we present our addition-only UPSI protocols. As briefly discussed in Section 1.2, each party stores an encrypted version of its set on the other party's storage. We first describe our new oblivious data structure maintained in a binary tree.

Oblivious Data Structure. Say P_1 is the data owner, who stores her encrypted set on P_0's side. Initially, the binary tree is empty with depth 0. Each node of the tree has a maximum capacity of σ elements. As P_1 adds new elements to the tree, she will gradually increase the tree depth. Figure 2 illustrates a tree of depth 3. Each element x is associated with a designated path computed by $F_{\mathsf{k}}(x)$, where F is a pseudorandom function and k is a secret key known to both parties. When a new element x is added to P_1's set, P_1 will add x to the one of the nodes in the root-to-leaf path ending at leaf node $F_{\mathsf{k}}(x)$, but in an oblivious way. In the example in Figure 2, the designated path of x is $F_{\mathsf{k}}(x) = 001$, and P_1 will obliviously add x to one of the four nodes on the red path. To do so, P_1 first adds x to the root node of the tree. Then she samples a random root-to-leaf path ℓ of the tree, and collects all the elements in that random path. For every element x^* in that random path (note that this includes x, because x was just added to the root), P_1 will "push down" x^* along the random path ℓ as much as possible subject to the constraint that x^* is still on its designated path $F_{\mathsf{k}}(x^*)$. In the example, $\ell = 011$, and P_1 considers all the elements on the blue path. She can push x down one level since it overlaps with the red path. For another element y, suppose $F_{\mathsf{k}}(y) = 011$, then P_1 can push it down to the leaf level. For the element z, suppose $F_{\mathsf{k}}(z) = 010$, then P_1 cannot push it down further. Note that this process is oblivious to P_0 since the access pattern for *any* element is a random path. In the example, the access pattern for x is a random path ℓ that is completely independent of x.

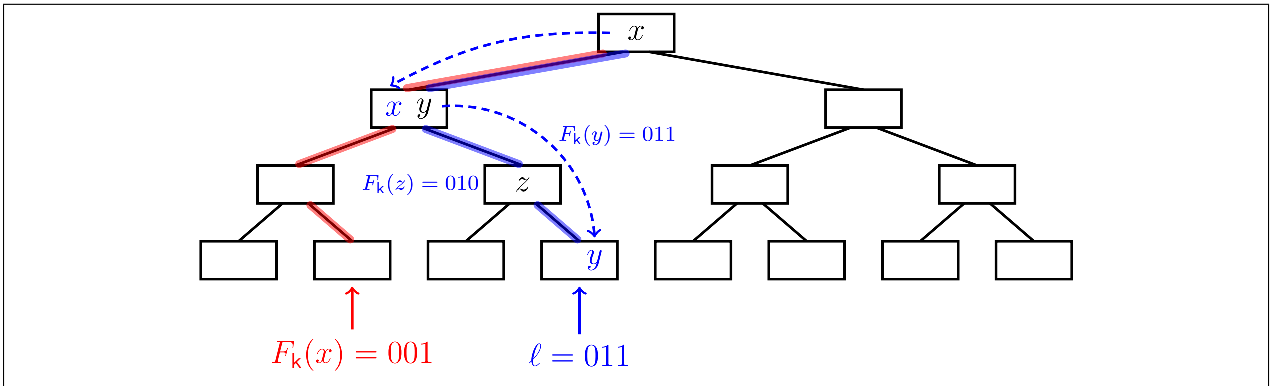

Figure 2: Illustration of adding an element x to a tree with depth 3.

Some details were omitted in the above description for the sake of simplicity. First, when pushing down element along the random path ℓ, another constraint is that no node exceeds the maximum capacity of σ. Second, if there are extra elements that cannot fit into the maximum capacity of the random path, P_1 puts them into a *stash*, which has maximum capacity ρ. Both σ and ρ are defined as part of the security parameters of the protocol. We present this subroutine formally as UpdateTree in Figure 3. This subroutine will also be used in our UPSI with both addition and deletion protocols, with slight modifications (highlighted in the figure). We discuss more details in Section 4.

Addition-Only UPSI-Cardinalty/Sum/Circuit-PSI. We now describe our new addition-only UPSI protocols (Figure 5). P_0 maintains his elements $x \in X$ in an oblivious data structure consisting of a binary tree $\mathcal{D}_0$ and a stash $\mathcal{S}_0$. He stores an encrypted version of it on P_1's side, denoted as $(\widetilde{\mathcal{D}}_0, \widetilde{\mathcal{S}}_0)$. Similarly, P_1 maintains her elements $y \in Y$ in an oblivious data structure $(\mathcal{D}_1, \mathcal{S}_1)$, and stores an encrypted version $(\widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1)$ on P_0's side. The encryption scheme is a $(2,2)$-threshold additively homomorphic encryption. Recall from Section 1.2 that the set difference $I_d \setminus I$ on each day consists of two disjoint sets, $(X_d \cap Y)$ and $((X \cup X_d) \cap Y_d)$.

Let's first consider $(X_d \cap Y)$. Intuitively speaking, P_0 queries each $x_i \in X_d$ in the encrypted tree of Y, namely $(\widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1)$, to determine whether $x_i \in Y$. Specifically, for each $x_i \in X_d$, P_0 identifies a designated path $\ell = F_{\mathsf{k}}(x_i)$ and collects all the elements in the path ℓ from $\widetilde{\mathcal{D}}_1$, together with all the elements from $\widetilde{\mathcal{S}}_1$ (because x_i could potentially have been put there as well). These are all the candidate encryptions that could potentially match x_i. This process is presented formally as a subroutine GetPath in Figure 4. To compute PSI-Cardinality, P_0 homomorphically subtracts x_i from each candidate encryption, so it becomes an encryption of zero iff it is a match. This is presented as Step 3 in Figure 5.

Symmetrically, for $((X \cup X_d) \cap Y_d)$, P_1 queries each $y_j \in Y_d$ in the encrypted tree of $(X \cup X_d)$, namely $(\widetilde{\mathcal{D}}_0, \widetilde{\mathcal{S}}_0)$. Note that $(\widetilde{\mathcal{D}}_0, \widetilde{\mathcal{S}}_0)$ needs to be first updated to contain X_d. In the protocol in Figure 4, P_0 adds X_d to the oblivious data structure in Step 3. Then P_1 collects all the candidate encryptions for each $y_j \in Y_d$ and homomorphically subtracts y_j from them, as presented in Step 4.

In Step 5, P_1 combines all the candidate encryptions and homomorphically multiplies each one by a random scalar, so that a candidate encryption remains zero if it is a match, or random

Subroutine $\mathsf{UpdateTree}(\{x_i\}_{i=1}^n, \{p_i\}_{i=1}^n, \mathcal{D}, \mathcal{S}, F_\mathsf{k}(\cdot), \mathsf{Enc}_\mathsf{pk}(\cdot))$:

1. Let N be the total number of elements (excluding dummy ones) in the tree $\mathcal{D}$ and stash $\mathcal{S}$ after inserting $\{x_i\}_{i=1}^n$. Extend the tree depth to reach $L = \lceil \log_2 N \rceil$ if needed. Add empty nodes in the new levels of $\mathcal{D}$.
2. For each element and payload pair (x_i, p_i) for $i \in [n]$:
   (a) Uniformly sample a random leaf node $\ell_i \xleftarrow{\$} \{0, 1, \ldots, 2^L - 1\}$ of the tree $\mathcal{D}$.
   (b) Remove all the elements from the path $\mathcal{P}(\ell_i)$ of the tree $\mathcal{D}$. Remove all the elements from the stash $\mathcal{S}$. Combine all the removed elements (excluding dummy ones) with (x_i, p_i) to get path_i. In the UPSI with addition and deletion protocols, if there are elements with opposite values, namely (z, p) and $(z, -p)$, then remove both from path_i.
   (c) For k from L down to 0:
   Consider the tree node $\mathcal{P}(\ell_i, k)$ at level k, remove up to σ elements (z, p) from path_i such that $\mathcal{P}(\ell_i, k) = \mathcal{P}(F_\mathsf{k}(z)_{[1:L]}, k)$, and add these elements to the node $\mathcal{P}(\ell_i, k)$ of $\mathcal{D}$.
   (d) Replace the stash $\mathcal{S}$ with all the elements left in path_i. If there are more than ρ elements left in path_i, abort.
   (e) Pad every node in the path $\mathcal{P}(\ell_i)$ with dummy elements to reach a size of σ. Pad the stash $\mathcal{S}$ with dummy elements to reach a size of ρ.
3. For each $i \in [n]$, gather all the elements in the path $\mathcal{P}(\ell_i)$ and encrypt them to get $\widetilde{\mathsf{updates}}_i = \{(\mathsf{Enc}_\mathsf{pk}(x_j), \mathsf{Enc}_\mathsf{pk}(p_j))\}_{j=1}^{\sigma \cdot L}$. Encrypt all elements in the stash $\mathcal{S}$ to get $\widetilde{\mathcal{S}} = \{(\mathsf{Enc}_\mathsf{pk}(x_j), \mathsf{Enc}_\mathsf{pk}(p_j))\}_{j=1}^{\rho}$. Output $(\{(\widetilde{\mathsf{updates}}_i, \ell_i)\}_{i=1}^n, \widetilde{\mathcal{S}})$

Figure 3: Subroutine UpdateTree that outputs a succinct update for the tree $\mathcal{D}$ that does not reveal the elements being added.

Subroutine $\mathsf{GetPath}(\widetilde{\mathcal{D}}, \widetilde{\mathcal{S}}, F_\mathsf{k}(\cdot), x)$:

1. Let L be the height of the tree $\mathcal{D}$.
2. Compute the leaf node for the path containing x as $\ell := F_\mathsf{k}(x)_{[1:L]}$.
3. Collect all the elements in the path $\mathcal{P}(\ell)$, combine them with the stash $\mathcal{S}$ to get $\widetilde{\mathsf{path}} = \{(\mathsf{Enc}(y_i), \mathsf{Enc}(p_i))\}_{i=1}^{\sigma \cdot L + \rho}$, and output $\widetilde{\mathsf{path}}$.

Figure 4: Subroutine GetPath that outputs a collection of potential matching elements with x in the encrypted tree $\widetilde{\mathcal{D}}$ with stash $\widetilde{\mathcal{S}}$ organized according to the pseudorandom function F.

otherwise.[2] She then randomly shuffles all the candidate encryptions, partially decrypts them, and sends to P_0, who can then fully decrypt them and count the number of zeros.

Finally, P_1 adds Y_d to her oblivious data structure in Step 7. It is important to note that the order of tree updates for X_d and Y_d is critical in the protocol. In particular, the tree update for $(\widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1)$ can only occur after Step 3 to prevent doubly counting in PSI-Cardinality.

We can extend the protocol to PSI-Sum and Circuit-PSI by attaching a payload to each element and leveraging additive homomorphism on these payloads.

Addition-Only Plain UPSI. For addition-only plain UPSI $\mathcal{F}_{\mathsf{UPSI\text{-}Add_{psi}}}$, we don't have to store two trees. Instead, we can simply plug our new oblivious data structure into the addition-only UPSI protocol [BMX22, $\Pi_{\mathsf{UPSI\text{-}add\text{-}one}}$] to achieve better concrete efficiency than the two-tree solution

[2] Note that this holds because the plaintext space for the encryption scheme is $\mathbb{Z}_q$ for a prime q.

Initialization:

1. P_0 and P_1 jointly setup public and secret keys for a (2, 2)-threshold additively homomorphic encryption scheme $(\mathsf{pk}, \mathsf{sk}_0, \mathsf{sk}_1) \leftarrow \mathsf{KeyGen}(1^\lambda)$ where P_0 receives $(\mathsf{pk}, \mathsf{sk}_0)$ and P_1 receives $(\mathsf{pk}, \mathsf{sk}_1)$. This can be done via a one-time secure two-party computation. The two parties agree on a randomly sampled PRF key $\mathsf{k} \xleftarrow{\$} \{0, 1\}^\lambda$.
2. P_0 and P_1 generate initial trees with only an empty root and stash: $(\mathcal{D}_0, \mathcal{S}_0, \widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1)$ and $(\widetilde{\mathcal{D}}_0, \widetilde{\mathcal{S}}_0, \mathcal{D}_1, \mathcal{S}_1)$, respectively.
3. Initialize $C_0 = 0$ in $\Pi_{\mathsf{UPSI\text{-}Add_{ca}}}$ and $\Pi_{\mathsf{UPSI\text{-}Add_{circuit}}}$, $C_0 = V_0 = 0$ in $\Pi_{\mathsf{UPSI\text{-}Add_{sum}}}$.

Day *d*: P_0 and P_1 hold $(\mathcal{D}_0, \mathcal{S}_0, \widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1)$ and $(\widetilde{\mathcal{D}}_0, \widetilde{\mathcal{S}}_0, \mathcal{D}_1, \mathcal{S}_1)$, respectively. Let L_0 be the tree height of $\mathcal{D}_0$ and $\widetilde{\mathcal{D}}_0$, and L_1 be the tree height of $\mathcal{D}_1$ and $\widetilde{\mathcal{D}}_1$. Both parties update L_0 and L_1 as they update the trees below. Let X, Y denote the two parties' sets at the end of the previous day, respectively.
P_0 holds a new input set X_d and P_1 holds a new input set Y_d. Let $n = |X_d|$ and $m = |Y_d|$. In $\Pi_{\mathsf{UPSI\text{-}Add_{sum}}}$, P_0 holds a value $v_i \in \mathbb{Z}_q$ associated with each element $x_i \in X_d$.

1. P_0 defines a payload for each element $x_i \in X_d$ depending on the functionality: $p_i = x_i$ in $\Pi_{\mathsf{UPSI\text{-}Add_{circuit}}}$, $p_i = v_i$ in $\Pi_{\mathsf{UPSI\text{-}Add_{sum}}}$, and no payload is needed in $\Pi_{\mathsf{UPSI\text{-}Add_{ca}}}$.
2. **X_d tree update.** P_0 computes $m_1 = (\{(\widetilde{\mathsf{updates}}_i, \ell_i)\}_{i=1}^n, \widetilde{\mathcal{S}}'_0) \leftarrow \mathsf{UpdateTree}(X_d, \{p_i\}_{i=1}^n, \mathcal{D}_0, \mathcal{S}_0, F_{\mathsf{k}}(\cdot), \mathsf{Enc}_{\mathsf{pk}}(\cdot))$, and sends it to P_1, who then replaces each path $\mathcal{P}(\ell_i)$ with $\widetilde{\mathsf{updates}}_i$ in $\widetilde{\mathcal{D}}_0$, and replaces $\widetilde{\mathcal{S}}_0$ with $\widetilde{\mathcal{S}}'_0$. Both parties update L_0 if needed.
3. **Candidates for $X_d \cap Y$.** For each $x_i \in X_d$, P_0 computes $\{\mathsf{Enc}_{\mathsf{pk}}(y_{i,j})\}_{j=1}^{\sigma \cdot L_1 + \rho} \leftarrow \mathsf{GetPath}(\widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1, F_{\mathsf{k}}(\cdot), x_i)$, homomorphically subtracts x_i, and attaches an encryption of p_i to get $\widetilde{\mathsf{path}}_i = \{(\mathsf{Enc}_{\mathsf{pk}}(y_{i,j} - x_i), \mathsf{Enc}_{\mathsf{pk}}(p_i))\}_{j=1}^{\sigma \cdot L_1 + \rho}$. Then P_0 sends $m_2 = \{\widetilde{\mathsf{path}}_i\}_{i=1}^n$ to P_1.
4. **Candidates for $(X \cup X_d) \cap Y_d$.** For each $y_j \in Y_d$, P_1 computes $\{(\mathsf{Enc}_{\mathsf{pk}}(x_{j,i}), \mathsf{Enc}_{\mathsf{pk}}(p_i))\}_{i=1}^{\sigma \cdot L_0 + \rho} \leftarrow \mathsf{GetPath}(\widetilde{\mathcal{D}}_0, \widetilde{\mathcal{S}}_0, F_{\mathsf{k}}(\cdot), y_j)$, and homomorphically subtracts y_j to get $\widetilde{\mathsf{path}}_j = \{(\mathsf{Enc}_{\mathsf{pk}}(x_{j,i} - y_j), \mathsf{Enc}_{\mathsf{pk}}(p_i))\}_{i=1}^{\sigma \cdot L_0 + \rho}$.
5. **Combining candidates.** P_1 combines $\{\widetilde{\mathsf{path}}_j\}_{j=1}^m$ with $\{\widetilde{\mathsf{path}}_i\}_{i=1}^n$ received from P_0, randomly samples a mask $\alpha_k \xleftarrow{\$} \mathbb{Z}_q$ for each element in the combined set, and samples a random permutation π over $[\Gamma]$ where $\Gamma = \sigma \cdot (n \cdot L_1 + m \cdot L_0) + \rho \cdot (n + m)$. Compute and send the following to P_0:
$$m_3 = \pi\left(\{(\mathsf{PartDec}_{\mathsf{sk}_1}(\alpha_k \odot \mathsf{Enc}_{\mathsf{pk}}(a_k - b_k)), \mathsf{ReRand}_{\mathsf{pk}}(\mathsf{Enc}_{\mathsf{pk}}(p_k)))\}_{k=1}^{\Gamma}\right).$$
6. **Output generation.** P_0 fully decrypts the first element in each tuple of m_3 to get $\alpha_k(a_k - b_k)$. Let $K = \{k \mid \alpha_k(a_k - b_k) = 0\}$.
   - In $\Pi_{\mathsf{UPSI\text{-}Add_{ca}}}$, P_0 outputs $C_d = C_{d-1} + |K|$.
   - In $\Pi_{\mathsf{UPSI\text{-}Add_{sum}}}$, P_0 computes $m_4 = \bigoplus_{k \in K} \mathsf{Enc}_{\mathsf{pk}}(p_k)$ and sends it to P_1. P_1 responds to P_0 with $m'_4 = \mathsf{PartDec}_{\mathsf{sk}_1}(m_4)$. P_0 fully decrypts it to get $V = \mathsf{FullDec}_{\mathsf{sk}_0}(m'_4)$, and outputs $V_d = V_{d-1} + V$.
   - In $\Pi_{\mathsf{UPSI\text{-}Add_{circuit}}}$, P_0 samples a random share $[\![z_k]\!]_0 \xleftarrow{\$} \mathbb{Z}_q$ for all $k \in K$, outputs $C_d = C_{d-1} + |K|$ and an updated share set with new random shares $\{[\![z_k]\!]_0\}_{k \in K}$. Additionally, P_0 computes and sends the following to P_1:
   $$m_4 = \{\mathsf{PartDec}_{\mathsf{sk}_0}(\mathsf{Enc}_{\mathsf{pk}}(p_k) \oplus \mathsf{Enc}_{\mathsf{pk}}(-[\![z_k]\!]_0))\}_{k \in K}.$$
   P_1 fully decrypts m_4 using sk_1 to get its shares $\{[\![z_k]\!]_1\}_{k \in K}$, and outputs $C_d = C_{d-1} + |K|$ and an updated share set with new random shares $\{[\![z_k]\!]_1\}_{k \in K}$.
7. **Y_d tree update.** P_1 computes $m_5 = (\{(\widetilde{\mathsf{updates}}_j, \ell_j)\}_{j=1}^m, \widetilde{\mathcal{S}}'_1) \leftarrow \mathsf{UpdateTree}(Y_d, \perp, \mathcal{D}_1, \mathcal{S}_1, F_{\mathsf{k}}(\cdot), \mathsf{Enc}_{\mathsf{pk}}(\cdot))$, and sends it to P_0, who then replaces each path $\mathcal{P}(\ell_j)$ with $\widetilde{\mathsf{updates}}_j$ in $\widetilde{\mathcal{D}}_1$, and replaces $\widetilde{\mathcal{S}}_1$ with $\widetilde{\mathcal{S}}'_1$. Both parties update L_1 if needed.

Figure 5: Protocols $\Pi_{\mathsf{UPSI\text{-}Add_{ca}}}, \Pi_{\mathsf{UPSI\text{-}Add_{sum}}}, \Pi_{\mathsf{UPSI\text{-}Add_{circuit}}}$ for one-sided addition-only UPSI functionalities $\mathcal{F}_{\mathsf{UPSI\text{-}Add_{ca}}}, \mathcal{F}_{\mathsf{UPSI\text{-}Add_{sum}}}, \mathcal{F}_{\mathsf{UPSI\text{-}Add_{circuit}}}$, respectively, with the differences among the three protocols highlighted.

and much lower worst-case complexity than [BMX22]. We present the protocol $\Pi_{\mathsf{UPSI\text{-}Add_{psi}}}$ in Appendix A.

3.3 Complexity, Correctness and Security

On each day d, let the entire set sizes of the two parties be N and M, respectively. Let the update set sizes be n and m, respectively. Then both the computation and communication complexity are $O(n \log M + m \log N)$, assuming σ and ρ are both $O(1)$. We state the theorem below and defer its proof to Appendix B.

Theorem 3.2. *Assuming Π is a secure $(2,2)$-threshold additively homomorphic encryption scheme, F is a pseudorandom function, the protocols $\Pi_{\textsf{UPSI-Add}_{ca}}, \Pi_{\textsf{UPSI-Add}_{sum}}, \Pi_{\textsf{UPSI-Add}_{circuit}}$ (Figure 5) securely realize the ideal functionalities $\mathcal{F}_{\textsf{UPSI-Add}_{ca}}, \mathcal{F}_{\textsf{UPSI-Add}_{sum}}, \mathcal{F}_{\textsf{UPSI-Add}_{circuit}}$ (Figure 1), respectively, against semi-honest adversaries.*

4 UPSI with Addition and Deletion

4.1 Definition

Let $X_{[D]} = \{(X_1^+, X_1^-), \ldots, (X_D^+, X_D^-)\}$ and $Y_{[D]} = \{(Y_1^+, Y_1^-), \ldots, (Y_D^+, Y_D^-)\}$ be the inputs for P_0 and P_1 after D days, respectively. Here, X_d^+ denotes the elements to be added to P_0's set on day d, and X_d^- denotes the elements to be deleted from P_0's set on day d; similarly, Y_d^+ and Y_d^- denote the elements to be added and deleted, respectively, for P_1 on day d. The ideal functionalities are defined in Figure 6. Note that for $\mathcal{F}_{\textsf{UPSI-Del}_{sum}}$, we achieve a slightly more general functionality than PSI-Sum as defined in [IKN+20, MPR+20] (which is the definition used in our addition-only protocol) in that our functionality does *not* have to reveal the cardinality C_d along with V_d. Let $\mathcal{F}_0$ be the output for P_0 for all functionalities. Note that we don't consider the Circuit-PSI functionality in this setting, so P_1 has no output in the definition.

Definition 4.1 (One-Sided UPSI with Addition and Deletion)**.** *A protocol Π is semi-honest secure with respect to ideal functionality $\mathcal{F} \in \{\mathcal{F}_{\textsf{UPSI-Del}_{psi}}, \mathcal{F}_{\textsf{UPSI-Del}_{ca}}, \mathcal{F}_{\textsf{UPSI-Del}_{sum}}\}$ if there exist PPT simulators Sim_0 and Sim_1 such that, for any $D \in \mathbb{N}^+$ and any inputs $(X_{[D]}, Y_{[D]})$,*

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

Notation. We use the same notation as in Section 3, except that instead of a $(2,2)$-threshold additively homomorphic encryption scheme, we use a plain additively homomorphic encryption scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ (see definition in Section 2) over plaintext space $\mathbb{Z}_q$.

4.2 Construction

In this section, we present our UPSI protocols with both addition and deletion. The oblivious data structure presented in Section 3.2 only supports adding new elements to the tree. We first discuss how to extend the construction to also allow for deletion of elements from the tree.

Oblivious Data Structure with Deletion. Recall that each element x is associated with a designated path $F_{\mathsf{k}}(x)$. When P_1 adds a new element x to the tree, she will first add x to the root node of the tree. Then she samples a random path of the tree and pushes down elements along

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

Day d:

- **Public Parameters**: For $\mathcal{F}_{\mathsf{UPSI\text{-}Del}_{\mathsf{psi}}}$, the number of additions and deletions performed each day: $|X_d^-|$, $|X_d^+|$, $|Y_d^-|$, $|Y_d^+|$.

  For $\mathcal{F}_{\mathsf{UPSI\text{-}Del}_{\mathsf{ca}}}$ and $\mathcal{F}_{\mathsf{UPSI\text{-}Del}_{\mathsf{sum}}}$, the *combined* number of additions and deletions performed each day: $|X_d^- \cup X_d^+|$ and $|Y_d^- \cup Y_d^+|$.

- **Inputs:**
  P_0 inputs an addition set $X_d^+ \subseteq \{0,1\}^*$ where $X_d^+ \cap X = \emptyset$ and a deletion set $X_d^- \subseteq X$. In $\mathcal{F}_{\mathsf{UPSI\text{-}Del}_{\mathsf{sum}}}$, X_d^+ includes a value associated with each set member (i.e., v_i is associated with $x_i \in X_d^+$).
  P_1 inputs an addition set $Y_d^+ \subseteq \{0,1\}^*$ where $Y_d^+ \cap Y = \emptyset$ and a deletion set $Y_d^- \subseteq Y$.

- **Update**: On receiving the inputs from both parties, the ideal functionality updates $X = (X \cup X_d^+) \setminus X_d^-$ and $Y = (Y \cup Y_d^+) \setminus Y_d^-$.

- **Output:**
  In $\mathcal{F}_{\mathsf{UPSI\text{-}Del}_{\mathsf{psi}}}$, P_0 learns the intersection $I_d = X \cap Y$.
  In $\mathcal{F}_{\mathsf{UPSI\text{-}Del}_{\mathsf{ca}}}$, P_0 learns the cardinality $C_d = |X \cap Y|$.
  In $\mathcal{F}_{\mathsf{UPSI\text{-}Del}_{\mathsf{sum}}}$, P_0 learns $V_d = \sum_{i:x_i \in X \cap Y} v_i$.

Figure 6: Ideal functionalities for one-sided UPSI with both addition and deletion: $\mathcal{F}_{\mathsf{UPSI\text{-}Del}_{\mathsf{psi}}}$, $\mathcal{F}_{\mathsf{UPSI\text{-}Del}_{\mathsf{ca}}}$, and $\mathcal{F}_{\mathsf{UPSI\text{-}Del}_{\mathsf{sum}}}$.

that random path as much as possible. To support deletion, P_1 first attaches a payload p to each element x. When x is added to P_1's set, she sets $p = +1$; when x is deleted from her set, she sets $p = -1$. Whenever an element x is added *or* deleted from her set, P_1 adds a new pair (x, p) to the tree following the exact same approach as described in UpdateTree (Figure 3). The only minor difference is that when pushing down elements along the random path, if both $(x, +1)$ and $(x, -1)$ appear in that path, P_1 removes both of them from the tree.

This modified UpdateTree process remains oblivious to P_0 because the access pattern for addition or deletion of elements continues to be a random path together with the stash. Note that since additions and deletions of the same element have the same designated path, there is a higher probability of stash overflow if we use the same parameters of maximum node capacity σ and maximum stash capacity ρ as in the addition-only setting, hence we need to increase both parameters for our new protocols. We discuss the parameter implications in the security proofs (Lemma D.1).

Computation on Encrypted Tree. To compute on the encrypted tree, we take a different approach from the addition-only protocols. When P_0 queries an element x in the encrypted tree of Y, namely $(\widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1)$, he can still identify the designated path $\ell = F_{\mathsf{k}}(x)$ and collect all the candidate encryptions using GetPath (Figure 4). However, there could be both $(\mathsf{Enc}(x), \mathsf{Enc}(+1))$ and $(\mathsf{Enc}(x), \mathsf{Enc}(-1))$ among these candidates. In case x was added and then deleted from tree, it should be indistinguishable to P_0 from the case where x was never added to the tree. We construct a subprotocol $\Pi_{\mathsf{CombinePath}}$ (Figure 7) for the two parties to jointly learn a secret share of whether x is in the path, namely the sum of the associated payloads p for all the $(\mathsf{Enc}(x), \mathsf{Enc}(p))$ pairs.

Specifically, for each candidate encryption $(\mathsf{Enc}(y_i), \mathsf{Enc}(p_i))$, P_0 first homomorphically computes $\mathsf{Enc}(y_i - x + \alpha_i)$ for a randomly sampled α_i and sends it to P_1, which can then be decrypted by

Subprotocol $\Pi_{\mathsf{CombinePath}}((x, p, \widetilde{\mathsf{path}}), \mathsf{sk})$

Public Parameters: a public key pk for the additively homomorphic encryption scheme Π, and k as the number of pairs in $\widetilde{\mathsf{path}}$.

Inputs: An Initiator inputs an element x, an associated payload p, and a potential matching elements in an encrypted collection $\widetilde{\mathsf{path}} = \{(\mathsf{Enc}_{\mathsf{pk}}(y_i), \mathsf{Enc}_{\mathsf{pk}}(q_i))\}_{i=1}^{k}$. A Responder inputs the secret key sk corresponding to pk.

Output: Initiator and Responder receive a secret share of $\sum_{i \in [k]: x = y_i} (p \cdot q_i)$ over $\mathbb{Z}_q$.

1. For each $i \in [k]$, Initiator samples random masks $\alpha_i, \beta_i \xleftarrow{\$} \mathbb{Z}_q$ and homomorphically computes the following:
$$\begin{aligned} \mathsf{req}_i &= (\mathsf{Enc}_{\mathsf{pk}}(y_i) \oplus \mathsf{Enc}_{\mathsf{pk}}(\alpha_i - x)) \\ m_{i,0} &= p \odot \mathsf{Enc}_{\mathsf{pk}}(q_i) \oplus \mathsf{Enc}_{\mathsf{pk}}(-\beta_i) \\ m_{i,1} &= \mathsf{Enc}_{\mathsf{pk}}(-\beta_i) \end{aligned}$$
2. Initiator sends the request set $\{\mathsf{req}_i\}_{i=1}^{k}$ to Responder.
3. Responder decrypts each request with sk to get $\{\gamma_i\}_{i=1}^{k}$.
4. For all $i \in [k]$, both parties invoke $\mathcal{F}_{\mathsf{lookup}}$, where Initiator inputs $(\alpha_i, m_{i,0}, m_{i,1})$ as Sender and Responder inputs γ_i as Receiver, from which Responder receives m_i. Responder then sets $[\![r_i]\!]_1 = \mathsf{Dec}_{\mathsf{sk}}(m_i)$. Initiator sets $[\![r_i]\!]_0 = \beta_i$.
5. Each party P_b ($b \in \{0, 1\}$) outputs $\sum_{i=1}^{k} [\![r_i]\!]_b$.

Figure 7: Subprotocol $\Pi_{\mathsf{CombinePath}}$ required for UPSI with addition and deletion.

Inputs: A Sender inputs (a, m_0, m_1) where $a \in \mathbb{Z}_q$ and (m_0, m_1) are two messages of equal length. A Receiver inputs $b \in \mathbb{Z}_q$.

Output: If $a = b$, then output m_0 to Receiver; otherwise output m_1 to Receiver.

Figure 8: Ideal functionality $\mathcal{F}_{\mathsf{lookup}}$ required for the subprotocol $\Pi_{\mathsf{CombinePath}}$.

P_1 to γ_i. Note that $\alpha_i = \gamma_i$ iff $y_i = x$. Next, our goal is to design a special equality testing protocol such that if $\alpha_i = \gamma_i$ (i.e., $y_i = x$), then the two parties obtain a secret share of p_i, otherwise they obtain a secret share of 0. To do so, P_0 homomorphically computes two ciphertexts $m_{i,0} = \mathsf{Enc}(p_i - \beta_i)$ and $m_{i,0} = \mathsf{Enc}(-\beta_i)$ for a randomly sampled β_i. Then the two parties invoke a special secure two-party computation protocol with functionality $\mathcal{F}_{\mathsf{lookup}}$. The functionality $\mathcal{F}_{\mathsf{lookup}}$ takes $(\alpha_i, m_{i,0}, m_{i,1})$ from P_0 and γ_i from P_1 as input. If $\alpha_i = \gamma_i$, then $\mathcal{F}_{\mathsf{lookup}}$ outputs $m_{i,0}$ to P_1; otherwise it outputs $m_{i,1}$ to P_1. Therefore, if $\alpha_i = \gamma_i$, then P_1 obtains $\mathsf{Enc}(p_i - \beta_i)$, which can be decrypted to $p_i - \beta_i$, thereby forming a secret share of p_i with the other share β_i held by P_0. If $\alpha_i \neq \gamma_i$, then P_1 obtains a $\mathsf{Enc}(-\beta_i)$, which can be decrypted to $-\beta_i$, forming a secret share of 0 with P_0's share β_i. As a result, the two parties obtain a secret share of p_i if $y_i = x$, or a secret share of 0 otherwise. Finally, the two parties sum up all the secret shares to obtain a secret share of $\sum_{y_i = x} p_i$.

We present our subprotocol $\Pi_{\mathsf{CombinePath}}$ in Figure 7 and defer its correctness and security proofs to Appendix C. The functionality $\mathcal{F}_{\mathsf{lookup}}$ can be instantiated with a generic secure two-party computation protocol [Yao86, GMW87]. We present a more efficient realization utilizing oblivious transfer (OT) and the efficient OT extension [IKNP03, ALSZ13] in Section 5.

UPSI-Cardinalty/Sum with Addition and Deletion. Next, we describe our new UPSI protocols with both addition and deletion for PSI-Cardinality and PSI-Sum, presented in Figure 9. To compute PSI-Cardinality, we follow the similar framework as in the addition-only protocols (Figure 5).

Initialization:

1. P_0 and P_1 independently generate key pairs for an additive homomorphic encryption scheme $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and share the public keys. Both parties agree on a randomly sampled PRF key $\mathsf{k} \xleftarrow{\$} \{0,1\}^\lambda$.
2. P_0 and P_1 generate initial trees with only an empty root and stash: $(\mathcal{D}_0, \mathcal{S}_0, \widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1)$ and $(\widetilde{\mathcal{D}}_0, \widetilde{\mathcal{S}}_0, \mathcal{D}_1, \mathcal{S}_1)$, respectively. Initialize $\mathsf{Out}_0 = 0$.

Day d: P_0 and P_1 hold $(\mathcal{D}_0, \mathcal{S}_0, \widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1)$ and $(\widetilde{\mathcal{D}}_0, \widetilde{\mathcal{S}}_0, \mathcal{D}_1, \mathcal{S}_1)$, respectively. Let L_0 and L_1 be the heights of $\mathcal{D}_0$ (and $\widetilde{\mathcal{D}}_0$), and $\mathcal{D}_1$ (and $\widetilde{\mathcal{D}}_1$) respectively. Both parties update L_0 and L_1 as they update the trees below. Let X, Y denote the two parties' sets at the end of the previous day.
P_0 and P_1 have new input sets X_d^+, Y_d^+ which include elements they are adding to their set and X_d^-, Y_d^- of elements they are deleting. Denote $n = |X_d^+ \cup X_d^-|$, $m = |Y_d^+ \cup Y_d^-|$. In $\Pi_{\mathsf{UPSI\text{-}Del_{sum}}}$, P_0 holds a value $v_i \in \mathbb{Z}_q$ associated with each element $x_i \in X_d^+ \cup X_d^-$.

1. P_0 defines a payload for each element $x_i \in X_d^+ \cup X_d^-$ depending on the functionality:
$$p_i := \begin{cases} (-1)^{(x_i \in X_d^-)} & \text{for } \Pi_{\mathsf{UPSI\text{-}Del_{ca}}} \\ (-1)^{(x_i \in X_d^-)} \cdot v_i & \text{for } \Pi_{\mathsf{UPSI\text{-}Del_{sum}}} \end{cases}$$
P_1 defines a payload for each element $y_i \in Y_d^+ \cup Y_d^-$: $q_j := (-1)^{(y_j \in Y_d^-)}$.
2. $\boldsymbol{X_d^+ \cup X_d^-}$ **tree update.** P_0 sends $(\{(\widetilde{\mathsf{updates}}_i, \ell_i)\}_{i=1}^n, \widetilde{\mathcal{S}}_0') \leftarrow \mathsf{UpdateTree}(X_d^+ \cup X_d^-, \{p_i\}_{i=1}^n, \mathcal{D}_0, \mathcal{S}_0, F_{\mathsf{k}}(\cdot), \mathsf{Enc}_{\mathsf{pk}_0}(\cdot))$ to P_1. P_1 replaces each path $\mathcal{P}(\ell_i)$ with $\widetilde{\mathsf{updates}}_i$ in $\widetilde{\mathcal{D}}_0$, and replaces $\widetilde{\mathcal{S}}_0$ with $\widetilde{\mathcal{S}}_0'$.
3. **Secret shares for new elements of $\boldsymbol{X}$.** For all $x_i \in (X_d^+ \cup X_d^-)$, run $\Pi_{\mathsf{CombinePath}}$ with P_0 as $\mathsf{Initiator}$ inputting $(x_i, p_i, \widetilde{\mathsf{path}}_i \leftarrow \mathsf{GetPath}(\widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1, F_{\mathsf{k}}(\cdot), x_i), \mathsf{pk}_1)$ and P_1 as $\mathsf{Responder}$ inputting sk_1 corresponding to pk_1. They receive secret shares $[\![z_{x,i}]\!]_0$ and $[\![z_{x,i}]\!]_1$, respectively.
4. **Secret shares for new elements of $\boldsymbol{Y}$.** For all $y_j \in (Y_d^+ \cup Y_d^-)$, run $\Pi_{\mathsf{CombinePath}}$ with P_0 as $\mathsf{Responder}$ inputting sk_0 corresponding to pk_0) and P_1 as $\mathsf{Initiator}$ inputting $(y_j, q_j, \mathsf{path}_j \leftarrow \mathsf{GetPath}(\widetilde{\mathcal{D}}_0, \widetilde{\mathcal{S}}_0, F_{\mathsf{k}}(\cdot), y_j), \mathsf{pk}_0)$. They receive secret shares $[\![z_{y,j}]\!]_0$ and $[\![z_{y,j}]\!]_1$, respectively.
5. $\boldsymbol{Y_d^+ \cup Y_d^-}$ **tree update.** P_1 sends $(\{(\widetilde{\mathsf{updates}}_j, \ell_j)\}_{j=1}^m, \widetilde{\mathcal{S}}_1') \leftarrow \mathsf{UpdateTree}(Y_d^+ \cup Y_d^-, \{q_j\}_{j=1}^m, \mathcal{D}_1, \mathcal{S}_1, F_{\mathsf{k}}(\cdot), \mathsf{Enc}_{\mathsf{pk}_1}(\cdot))$ to P_0. P_0 replaces each path $\mathcal{P}(\ell_j)$ with $\widetilde{\mathsf{updates}}_j$ in $\widetilde{\mathcal{D}}_1$, and replaces $\widetilde{\mathcal{S}}_1$ with $\widetilde{\mathcal{S}}_1'$.
6. **Combine all the shares.** For $b \in \{0,1\}$, P_b computes $[\![z_d]\!]_b := \sum_{i=1}^n [\![z_{x,i}]\!]_b + \sum_{j=1}^m [\![z_{y,j}]\!]_b$.
7. **Output generation:** P_1 sends $[\![z_d]\!]_1$ to P_0, who then computes $\mathsf{Out}_d := \mathsf{Out}_{d-1} + [\![z_d]\!]_0 + [\![z_d]\!]_1$.
P_0 outputs Out_d for both $\Pi_{\mathsf{UPSI\text{-}Del_{ca}}}$ and $\Pi_{\mathsf{UPSI\text{-}Del_{sum}}}$.

Figure 9: Protocols $\Pi_{\mathsf{UPSI\text{-}Del_{ca}}}$ and $\Pi_{\mathsf{UPSI\text{-}Del_{sum}}}$ for one-side UPSI with both addition and deletion functionalities $\mathcal{F}_{\mathsf{UPSI\text{-}Del_{ca}}}$ and $\Pi_{\mathsf{UPSI\text{-}Del_{sum}}}$, respectively, with differences between the two protocols highlighted.

In Step 1, if the element x_i is deleted from the set, the payload p_i should be -1 for $\Pi_{\mathsf{UPSI\text{-}Del_{ca}}}$, and $-v_i$ for $\Pi_{\mathsf{UPSI\text{-}Del_{sum}}}$. If the element x_i is added to the set, the payload p_i should be $+1$ for $\Pi_{\mathsf{UPSI\text{-}Del_{ca}}}$, and v_i for $\Pi_{\mathsf{UPSI\text{-}Del_{sum}}}$. In Step 2, P_0 adds all the elements in $X_d^+ \cup X_d^-$ to his tree using the oblivious data structure with deletion. In Step 3, P_0 queries each element $x_i \in X_d^+ \cup X_d^-$ in the encrypted tree of Y. For an element $x_i \in X_d^+$ to be added to the set, the two parties run

$\Pi_{\mathsf{CombinePath}}$ to get a secret share of whether $x_i \in Y$. For an element $x_i \in X_d^-$ to be deleted from the set, they need to slightly modify $\Pi_{\mathsf{CombinePath}}$ to get a secret share of $(-1) \cdot$ (whether $x_i \in Y$). This means x_i was in the intersection but deleted from P_0's set in this step, so PSI-Cardinality is decreased by 1. In our protocol for $\Pi_{\mathsf{CombinePath}}$ (Figure 7), P_0 inputs an additional value (+1 or −1) to be multiplied with the result, which is done homomorphically in the protocol. Symmetrically, P_1 queries each element $y_j \in X_d^+ \cup X_d^-$ in the encrypted tree of $(X \cup X_d^+) \setminus X_d^-$ in Step 4. After this, P_1 adds all the elements in $Y_d^+ \cup Y_d^-$ to her tree in Step 5 (recall that it must occur after Step 3).

Finally, the two parties add up all the secret shares in Step 6 and reveal the output in Step 7. This protocol can be naturally extended to PSI-Sum if P_0 attaches payloads of value $+v_i$ or $-v_i$ for each element x_i in $\mathsf{UpdateTree}$ and $\Pi_{\mathsf{CombinePath}}$. It is worth noting that parties only aggregate their secret shares at the end of the protocol, hence our PSI-Sum protocol does *not* have to reveal the cardinality of the intersection, which may be useful in certain applications.

Plain UPSI with Addition and Deletion. Interestingly, achieving plain UPSI is more challenging than PSI-Cardinality and PSI-Sum with addition and deletion. As briefly discussed in Section 1.2, one issue comes from the scenario when an element x is added by one party while being deleted by the other party on the same day. In our UPSI-Cardinality/Sum protocols, while adding and deleting x from the intersection both occur on the same day, their effect on the output cancels out when their secret shares are combined. However, in plain UPSI, parties need to learn the exact elements to be added or deleted. Revealing that x was first added and then deleted from the intersection on the same day discloses more information than the ideal functionality.

To address this issue, we carefully arranged the sequence of the addition and deletion operations, as presented in Figure 10, such that deletions are dealt with in Step 1 before additions in Step 2. In other words, if x is deleted by P_0 while being added by P_1 on the same day, it will be first deleted from P_0's tree, so that it won't appear in the intersection when P_1 queries x in the encrypted tree. Since additions and deletions are done separately, both parties need to know $|X_d^-|$, $|X_d^+|$, $|Y_d^-|$, $|Y_d^+|$ on each day. This is different from UPSI-Cardinality/Sum where they only know $|X_d^- \cup X_d^+|$ and $|Y_d^- \cup Y_d^+|$, as reflected in the ideal functionalities (Figure 6).

Furthermore, unlike UPSI-Cardinality/Sum where parties sum up all the secret shared results at the end of the protocol, they need to learn the results for each individual element in plain UPSI. However, they cannot reveal directly these results because doing so may disclose more information than the ideal functionality. Specifically, if an element x is deleted from both sets on the same day (hence deleted from the intersection), our protocol ensures that the deleted x only appears once in either Step 1b or Step 1c, but it should be hidden from the parties whether the other party also deleted x on that day. To achieve this, the parties re-randomize and shuffle the results in Step 3.

4.3 Complexity, Correctness and Security

UPSI-Cardinalty/Sum with Addition and Deletion. Our protocols for $\Pi_{\mathsf{UPSI\text{-}Del_{ca}}}$ and $\Pi_{\mathsf{UPSI\text{-}Del_{sum}}}$ are presented in Figure 9. On each day d, let N, M be the total number of additions and deletions of the two parties, respectively. Let the update set sizes be n and m, respectively. Then both the computation and communication complexity are $O(n \cdot (\sigma \cdot \log M + \rho) + m \cdot (\sigma \cdot \log N + \rho))$. We state the theorem below and defer its proof to Appendix E.

Theorem 4.2. *Assuming* Π *is a secure additively homomorphic encryption scheme,* F *is a pseudorandom function, the protocols* $\Pi_{\mathsf{UPSI\text{-}Del_{ca}}}, \Pi_{\mathsf{UPSI\text{-}Del_{sum}}}$ *presented in Figure 9 securely realize the*

Initialization:

1. P_0 and P_1 independently generate key pairs for an additive homomorphic encryption scheme $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and share the public keys. Both parties agree on a randomly sampled PRF key $\mathsf{k} \xleftarrow{\$} \{0,1\}^\lambda$.
2. P_0 and P_1 generate initial trees with only an empty root and stash: $(\mathcal{D}_0, \mathcal{S}_0, \widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1)$ and $(\widetilde{\mathcal{D}}_0, \widetilde{\mathcal{S}}_0, \mathcal{D}_1, \mathcal{S}_1)$, respectively. Initialize $I_0 = \emptyset$.

Day d: P_0 and P_1 hold $(\mathcal{D}_0, \mathcal{S}_0, \widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1)$ and $(\widetilde{\mathcal{D}}_0, \widetilde{\mathcal{S}}_0, \mathcal{D}_1, \mathcal{S}_1)$, respectively. Let L_0 and L_1 be the heights of $\mathcal{D}_0$ (and $\widetilde{\mathcal{D}}_0$), and $\mathcal{D}_1$ (and $\widetilde{\mathcal{D}}_1$) respectively. Both parties update L_0 and L_1 as they update the trees below. Let X, Y denote the two parties' sets at the end of the previous day.
P_0 and P_1 have new input sets X_d^+, Y_d^+ which include elements they are adding to their set and X_d^-, Y_d^- of elements they are deleting. Denote $n^- = |X_d^-|$, $n^+ = |X_d^+|$, $m^- = |Y_d^-|$, $m^+ = |Y_d^+|$.

1. **Deletion:**
   (a) $\boldsymbol{X_d^-}$ **tree update.** P_0 sends $(\{(\widetilde{\mathsf{updates}}_i, \ell_i)\}_{i=1}^{n^-}, \widetilde{\mathcal{S}'_0}) \leftarrow \mathsf{UpdateTree}(X_d^-, \{-x_i : x_i \in X_d^-\}_{i=1}^{n^-}, \mathcal{D}_0, \mathcal{S}_0, F_\mathsf{k}(\cdot), \mathsf{Enc}_{\mathsf{pk}_0}(\cdot))$ to P_1. P_1 replaces each path $\mathcal{P}(\ell_i)$ with $\widetilde{\mathsf{updates}}_i$ in $\widetilde{\mathcal{D}}_0$, and replaces $\widetilde{\mathcal{S}}_0$ with $\widetilde{\mathcal{S}}'_0$.

   (b) **Secret shares for $\boldsymbol{X_d^- \cap Y}$.** For all $x_i \in X_d^-$, run $\Pi_{\mathsf{CombinePath}}$ with P_0 as $\mathsf{Initiator}$ inputting $(x_i, -1, \widetilde{\mathsf{path}}_i \leftarrow \mathsf{GetPath}(\widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1, F_\mathsf{k}(\cdot), x_i))$ and P_1 as $\mathsf{Responder}$ inputting sk_1 corresponding to pk_1. They receive secret shares $[\![z_{x,i}^-]\!]_0$ and $[\![z_{x,i}^-]\!]_1$, respectively, where $z_{x,i}^- = -x_i$ if $x_i \in \mathcal{D}_1 \cup \mathcal{S}_1$ and 0 otherwise.

   (c) **Secret shares for $\boldsymbol{\left(X \setminus X_d^-\right) \cap Y_d^-}$.** For all $y_j \in Y_d^-$, run $\Pi_{\mathsf{CombinePath}}$ with P_0 as $\mathsf{Responder}$ inputting sk_0 corresponding to pk_1 and P_1 as $\mathsf{Initiator}$ inputting $(y_j, -1, \widetilde{\mathsf{path}}_j \leftarrow \mathsf{GetPath}(\widetilde{\mathcal{D}}_0, \widetilde{\mathcal{S}}_0, F_\mathsf{k}(\cdot), y_j))$. They receive secret shares $[\![z_{y,j}^-]\!]_0$ and $[\![z_{y,j}^-]\!]_1$, respectively, where $z_{y,j}^- = -y_j$ if $y_j \in \mathcal{D}_0 \cup \mathcal{S}_0$ and 0 otherwise.

   (d) $\boldsymbol{Y_d^-}$ **tree update.** P_1 sends $(\{(\widetilde{\mathsf{updates}}_j, \ell_j)\}_{j=1}^{m^-}, \widetilde{\mathcal{S}'_1}) \leftarrow \mathsf{UpdateTree}(Y_d^-, \{-y_j : y_j \in Y_d^-\}_{j=1}^{m^-}, \mathcal{D}_1, \mathcal{S}_1, F_\mathsf{k}(\cdot), \mathsf{Enc}_{\mathsf{pk}_1}(\cdot))$ to P_0. P_0 replaces each path $\mathcal{P}(\ell_j)$ with $\widetilde{\mathsf{updates}}_j$ in $\widetilde{\mathcal{D}}_1$, and replaces $\widetilde{\mathcal{S}}_1$ with $\widetilde{\mathcal{S}}'_1$.

2. **Addition:**
   (a) $\boldsymbol{X_d^+}$ **tree update.** P_0 sends $(\{(\widetilde{\mathsf{updates}}_i, \ell_i)\}_{i=1}^{n^+}, \widetilde{\mathcal{S}'_0}) \leftarrow \mathsf{UpdateTree}(X_d^+, \{x_i : x_i \in X_d^+\}_{i=1}^{n^+}, \mathcal{D}_0, \mathcal{S}_0, F_\mathsf{k}(\cdot), \mathsf{Enc}_{\mathsf{pk}_0}(\cdot))$ to P_1. P_1 replaces each path $\mathcal{P}(\ell_i)$ with $\widetilde{\mathsf{updates}}_i$ in $\widetilde{\mathcal{D}}_0$, and replaces $\widetilde{\mathcal{S}}_0$ with $\widetilde{\mathcal{S}}'_0$.

   (b) **Secret shares for $\boldsymbol{X_d^+ \cap \left(Y \setminus Y_d^-\right)}$.** For all $x_i \in X_d^+$, run $\Pi_{\mathsf{CombinePath}}$ with P_0 as $\mathsf{Initiator}$ inputting $(x_i, 1, \widetilde{\mathsf{path}}_i \leftarrow \mathsf{GetPath}(\widetilde{\mathcal{D}}_1, \widetilde{\mathcal{S}}_1, F_\mathsf{k}(\cdot), x_i))$ and P_1 as $\mathsf{Responder}$ inputting sk_1 corresponding to pk_1. They receive secret shares $[\![z_{x,i}^+]\!]_0$ and $[\![z_{x,i}^+]\!]_1$, respectively, where $z_{x,i}^+ = x_i$ if $x_i \in \mathcal{D}_1 \cup \mathcal{S}_1$ and 0 otherwise.

   (c) **Secret shares for $\boldsymbol{\left(X \cup X_d^+ \setminus X_d^-\right) \cap Y_d^+}$.** For all $y_j \in Y_d^+$, run $\Pi_{\mathsf{CombinePath}}$ with P_0 as $\mathsf{Responder}$ inputting sk_0 corresponding to pk_0 and P_1 as $\mathsf{Initiator}$ inputting $(y_j, 1, \widetilde{\mathsf{path}}_j \leftarrow \mathsf{GetPath}(\widetilde{\mathcal{D}}_0, \widetilde{\mathcal{S}}_0, F_\mathsf{k}(\cdot), y_j))$. They receive secret shares $[\![z_{y,j}^+]\!]_0$ and $[\![z_{y,j}^+]\!]_1$, respectively, where $z_{y,j}^+ = y_j$ if $y_j \in \mathcal{D}_0 \cup \mathcal{S}_0$ and 0 otherwise.

   (d) $\boldsymbol{Y_d^+}$ **tree update.** P_1 sends $(\{(\widetilde{\mathsf{updates}}_j, \ell_j)\}_{j=1}^{m^-}, \widetilde{\mathcal{S}'_1}) \leftarrow \mathsf{UpdateTree}(Y_d^+, \{y_j : y_j \in Y_d^+\}_{j=1}^{m^+}, \mathcal{D}_1, \mathcal{S}_1, F_\mathsf{k}(\cdot), \mathsf{Enc}_{\mathsf{pk}_1}(\cdot))$ to P_0. P_0 replaces each path $\mathcal{P}(\ell_j)$ with $\widetilde{\mathsf{updates}}_j$ in $\widetilde{\mathcal{D}}_1$, and replaces $\widetilde{\mathcal{S}}_1$ with $\widetilde{\mathcal{S}}'_1$.

3. **Output Generation:**
   (a) Let $\{[\![z_i]\!]_0\}_{i=1}^{\Gamma}$ and $\{[\![z_i]\!]_1\}_{i=1}^{\Gamma}$ be the shares received by P_0 and P_1 above, where $\Gamma = n^- + m^- + n^+ + m^+$. P_0 sends $\{\mathsf{Enc}_{\mathsf{pk}_0}([\![z_i]\!]_0)\}_{i=1}^{\Gamma}$ to P_1.

   (b) P_1 samples a random permutation π over $[\Gamma]$. P_1 samples a random mask $\alpha_i \xleftarrow{\$} \mathbb{Z}_q$ for each $i \in [\Gamma]$ and homomorphically adds them to the encryptions received from P_0. P_1 sends the following to P_0: $\pi\left(\left\{(\mathsf{Enc}_{\mathsf{pk}_0}([\![z_i]\!]_0) \oplus \mathsf{Enc}_{\mathsf{pk}_0}(\alpha_i)), [\![z_i]\!]_1 - \alpha_i)\right\}_{i=1}^{\Gamma}\right)$.

   (c) P_0 decrypts the first element in each pair using sk_0, and adds up each pair of shares to learn the shuffled set $\{z_j\}_{j=1}^{\Gamma}$.
   Output $I_d := I_{d-1} \cup \{z_j | z_j > 0\} \setminus \{-z_j | z_j < 0\}$.

Figure 10: Protocol $\Pi_{\mathsf{UPSI\text{-}Del_{psi}}}$ for one-sided UPSI with addition and deletion functionality $\mathcal{F}_{\mathsf{UPSI\text{-}Del_{psi}}}$.

ideal functionalities $\mathcal{F}_{\textsf{UPSI-Del}_{ca}}, \mathcal{F}_{\textsf{UPSI-Del}_{sum}}$ *defined in Figure 6, respectively, against semi-honest adversaries.*

Plain UPSI with Addition and Deletion. We present our protocol $\Pi_{\textsf{UPSI-Del}_{\textsf{psi}}}$ in Figure 10. On each day d, let N, M be the total number of additions and deletions of the two parties, respectively. Let the update set sizes be n and m, respectively. Then both the computation and communication complexity are $O(n \cdot (\sigma \cdot \log M + \rho) + m \cdot (\sigma \cdot \log N + \rho))$. We state the theorem below and defer its proof to Appendix D.

Theorem 4.3. *Assuming* Π *is a secure additively homomorphic encryption scheme,* F *is a pseudorandom function, the protocol* $\Pi_{\textsf{UPSI-Del}_{psi}}$ *presented in Figure 10 securely realizes the ideal functionalities* $\mathcal{F}_{\textsf{UPSI-Del}_{psi}}$ *defined in Figure 6 against semi-honest adversaries.*

5 Implementation Details and Optimizations

In this section, we discuss instantiations of the building blocks in our UPSI protocols and optimizations to further improve the concrete efficiency.

Encryption Schemes. In the addition-only UPSI protocols $\Pi_{\textsf{UPSI-Add}_{\textsf{ca}}}$ and $\Pi_{\textsf{UPSI-Add}_{\textsf{sum}}}$, we instantiate the $(2, 2)$-threshold additively homomorphic encryption scheme with exponential El Gamal encryption [ElG85] to take advantage of efficient elliptic curve operations. Recall that in this scheme, $\textsf{Enc}(m) = (g^r, h^r \cdot g^m)$ where the public key consists of a group generator g and a random group element $h = g^s$ with a secret key s. In the $(2, 2)$-threshold scheme, $\textsf{sk}_0$ and $\textsf{sk}_1$ form an additive secret share of s. Decryption of exponential El Gamal requires computing the discrete logarithm of a group element g^m, which is possible for a bounded message space. In all our addition-only UPSI protocols presented in Figure 5, decryption occurs in Step 6. Observe that P_0 does *not* have to fully decrypt the first element in each tuple of m_3; instead, it is sufficient to check whether the decrypted message is 0 or not. In particular, given a partially decrypted ciphertext $\hat{c} = (a, b)$, P_0 can determine if the encrypted message is 0 by checking if $b = a^{\textsf{sk}_0}$, without performing discrete logarithm. In $\Pi_{\textsf{UPSI-Add}_{\textsf{sum}}}$, P_0 needs to fully decrypt m'_4, where the underlying message can be bounded by the maximum sum of associated values.

In $\Pi_{\textsf{UPSI-Add}_{\textsf{circuit}}}$, while exponential El Gamal can still be used for the first ciphertext in m_3, the (masked) payload messages are distributed uniformly over the entire plaintext space, hence the payload messages are encrypted using $(2, 2)$-threshold Paillier encryption [Pai99] instead.

In our protocols with both addition and deletion presented in Section 4 ($\Pi_{\textsf{UPSI-Del}_{\textsf{psi}}}$ in Figure 10 and $\Pi_{\textsf{UPSI-Del}_{\textsf{ca}}}, \Pi_{\textsf{UPSI-Del}_{\textsf{sum}}}$ in Figure 9), El Gamal cannot be utilized because all the ciphertexts are encrypting secret shares that are distributed across the message space. Instead, the additively homomorphic encryption scheme is instantiated with Paillier. This has an impact on the computation time, as can be seen in Section 6.

Paillier Modulus Switching. Using Paillier in the deletion protocols introduces an additional technical challenge. Recall that the plaintext space in Paillier encryption is $\mathbb{Z}_n$ for a public key n, which is different for P_0's and P_1's keys. During our deletion protocols, parties perform $\Pi_{\textsf{CombinePath}}$ for both $\textsf{pk}_0 = n_0$ (P_0's public key) and $\textsf{pk}_1 = n_1$ (P_1's public key) to get secret shares in both $\mathbb{Z}_{n_0}$ and $\mathbb{Z}_{n_1}$. We discuss how to combine these secret shares over different moduli.

Let ℓ be the maximum bit length required to represent a set element or associated value. Recall that if set elements are of arbitrary length, we can apply a hash function on all the elements and perform PSI on the hash outputs. In our evaluation section, each party holds at most 2^{22} elements, hence there are at most 2^{23} total elements. If we model the hash function as a random oracle, to ensure collision probability lower than $2^{-\kappa}$ for statistically security parameter $\kappa = 40$, it is safe to bound $\ell = 85$. Let n be a Paillier public key and L be the bit length of n, which is typically 1536 or 2048.

Consider a value $r \in \mathbb{Z}_{2^\ell}$ being secret shared as $[\![r]\!]_0, [\![r]\!]_1 \in \mathbb{Z}_n$. We will convert this secret share into another secret share of r in $\mathbb{Z}_{2^\ell}$. First, the integer summation of $[\![r]\!]_0 + [\![r]\!]_1$ is either r or $r + n$, and the probability $\Pr[[\![r]\!]_0 + [\![r]\!]_1 = r] \leq \Pr[[\![r]\!]_0 \leq r] \leq 2^{\ell - L} \ll 2^{-\kappa}$. Therefore, with overwhelming probability $[\![r]\!]_0 + [\![r]\!]_1 = r + n$. Let $s_0 = [\![r]\!]_0$ and $s_1 = [\![r]\!]_1 - n$, then $s_0 + s_1 = r$, where $s_0 > 0$ and $s_1 < 0$ as integers. If we represent s_1 in two's complement format, then the lowest ℓ bits of $s_0 + s_1$ should be r and the higher order bits should all be 0. Therefore, we can take the ℓ lowest order bits of s_0 and s_1 (in two's complement format) to form a secret share of r in $\mathbb{Z}_{2^\ell}$. Given that the original secret shares $[\![r]\!]_0, [\![r]\!]_1 \in \mathbb{Z}_n$ are distributed randomly over $\mathbb{Z}_n$, the new shares are statistically close to a uniform distribution over $\mathbb{Z}_{2^\ell}$ because $\ell \ll L$.

Realizing $\mathcal{F}_{\mathsf{lookup}}$. While $\mathcal{F}_{\mathsf{lookup}}$ can be instantiated with a generic secure two-party computation (2PC) protocol [Yao86, GMW87], we construct a protocol that achieves better concrete efficiency, leveraging oblivious transfer (OT) and the efficient OT extension [IKNP03, ALSZ13]. Let (a, m_0, m_1) and b be the inputs to $\mathcal{F}_{\mathsf{lookup}}$ where m_0 is output when $a = b$ and m_1 otherwise. Before comparison, both parties compute a hash function $H : \mathbb{Z}_q \to \{0,1\}^{\ell_{\mathsf{gc}}}$ on their inputs a and b. The parties then run a garbled-circuit based equality testing to compute a binary secret share $[\![c]\!] \in \{0,1\}$ of $H(a) \stackrel{?}{=} H(b)$. Then two two parties run an OT protocol where Sender inputs two messages $(m_{1-[\![c]\!]_0}, m_{[\![c]\!]_0})$ and Receiver inputs a choice bit $[\![c]\!]_1$. If $a = b$, then $[\![c]\!]_0 \neq [\![c]\!]_1$, in which case Receiver will receive m_0, as desired in $\mathcal{F}_{\mathsf{lookup}}$; if $a \neq b$, then $[\![c]\!]_0 = [\![c]\!]_1$ with overwhelming probability (see analysis below), and the Responder will receive m_1.

In this approach, we need the guarantee that if $a \neq b$, then $H(a) \neq H(b)$ with overwhelming probability, hence ℓ_{gc} should be sufficiently large. On the other hand, the size of the equality testing circuit grows with ℓ_{gc}, so we want to choose the smallest ℓ_{gc} such that the probability of a failure (i.e., that $H(a) = H(b)$ for $a \neq b$) over the entire protocol is less than $2^{-\kappa}$. In all the benchmarks presented in Section 6, there are at most 2^{23} elements held by both parties, and each element is compared against at most 2^9 elements in $\Pi_{\mathsf{CombinePath}}$. Hence the total number of $\mathcal{F}_{\mathsf{lookup}}$ invocations is bounded by $2^{23} \cdot 2^9 = 2^{32}$. The overall failure probability is no greater than $2^{32} \cdot 2^{-\ell_{gc}}$, and we want to ensure statistical security, namely $2^{32} \cdot 2^{-\ell_{gc}} \leq 2^{-\kappa}$ for $\kappa = 40$. Therefore, we set $\ell_{gc} \approx 32 + 40 = 72$.

6 Evaluation

6.1 Experimental Setup

We implement all of our UPSI protocols in C++ and report their performance in this section. We use the crypto library as part of Google's open-sourced Private Join and Compute project [PJC] for El Gamal and Paillier encryptions, Google's `gRPC` [gRP] for networking, and `emp-tool` [WMK16] for instantiations of garbled circuits and oblivious transfer (including OT extension). Benchmarks are run on a Google Cloud [clo] `c2-standard-16` virtual machine with 64 GB of RAM. Each party is executed on a single thread and communicate over `localhost`. The Linux `tc` command is used to simulate the various network settings. We simulate the LAN connection with 0.2 ms RTT network latency and 1Gbps network bandwidth. For WAN connection, we set the RTT latency to be 80 ms and test on various network bandwidths including 200 Mbps, 50 Mbps, and 5 Mbps.

Addition-Only UPSI. To demonstrate the updatable property of our protocols, we consider the setting where both parties begin with an empty set to which N_d elements are added each day. Our benchmarks represent the performance of the protocols on day $(\frac{N}{N_d})$ where the size of each party's set reaches N.

We compare our plain UPSI protocols with the state-of-the-art semi-honest PSI protocol [RR22] (RR22), and compare our UPSI for extended functionalities (PSI-Cardinality, PSI-Sum, and Circuit-PSI) with the state-of-the-art Circuit-PSI [CGS22] (CGS22) and [RR22] (RR22), where, on day $(\frac{N}{N_d})$, the parties run PSI or Circuit-PSI on their full input sets of size N. Note that the Circuit-PSI protocols [CGS22, RR22] are also state-of-the-art for computing PSI-Cardinality or PSI-Sum, with slight modifications to their protocols. In our comparison, we assume these modifications do not incur extra overhead in their performance. We also compare our addition-only plain UPSI with [BMX22] to demonstrate the improvement of worst-case complexity by plugging in our new oblivious data structure.

We don't compare with the protocols specifically designed for PSI-Cardinality or PSI-Sum [IKN+20, GMR+21] because these protocols are outperformed by [CGS22,RR22]. A more recent work [BPSY23] improves PSI and Circuit-PSI communication by 12% compared to [RR22], but we don't compare with it for three reasons: (1) their construction is built on the Silver codes [CRR21], which turns out to be insecure [RRT23], (2) their source code is not available online, and (3) even if their construction is instantiated with secure codes, from our comparison with the other works, we expect our protocols to perform better in certain settings as well. Note that [RR22] is also instantiated with the insecure Silver codes, but their open-sourced library [RR] supports instantiating the construction with the state-of-the-art OT extension from expand-accumulate codes [BCG+22], which is what we compare with.

UPSI with Addition and Deletion. In the setting with both addition and deletion, standalone PSI protocols need only compute over elements that remain in the input sets. In the extreme case where the every element is added and then soon deleted, the input sets remain small and so the standalone PSI protocols would likely be optimal. Alternatively, if the input sets are growing at a steady rate, then our constructions may be best. These caveats should be understood and application-specific context would play a role in choosing a solution.

In our benchmarks, we assume roughly 3/4 set operations are additions and 1/4 are deletions. We further assume that each element can only be added and deleted *at most once* in each party's set (i.e., an element cannot be re-added once it has been deleted). In this case, the computation and communication complexity of our protocols are $O(N_d \cdot \log N)$.

Choice of N and N_d. In all of our experiments, we chose the values for N and N_d that would best demonstrate the *turning point* where we become competitive. Our protocols have more advantages when increasing the gap between N and N_d. As N increases (e.g., for billion-sized sets [KMRS14, BKC+23]), we expect our protocol to be dominant for more network settings and larger N_d values.

In all of our comparison tables, cells in green indicate the state-of-the-art performance, and those in blue indicate that our protocols perform better.

Concrete Parameters. We set the computational security parameter $\lambda = 128$ and the statistical security parameter $\kappa = 40$. Following the analysis in [SvS+13], we set the maximum tree node capacity $\sigma = 4$ and the maximum stash capacity $\rho = 89$ to achieve failure probability of 2^{-80} for inserting a single element into the tree. Even with our largest set size of 2^{22}, the combined failure probability is bounded well below $2^{-\kappa}$. In protocols with addition and deletion, we allow parties to add and delete each element at most once, and so we double both our node size (to $\sigma = 8$) and stash size (to $\rho = 178$) following Lemma D.1. To enable P_0 to efficiently decrypt m'_4 in Step 6 of $\Pi_{\textsf{UPSI-Add}_{\textsf{sum}}}$ (Figure 5) with exponential El Gamal encryption, we bound the PSI-Sum maximum value to be at most 10,000. Larger sums can either utilize extra storage with a lookup table or switch to using Paillier encryption. .

6.2 Addition-Only UPSI with Extended Functionalities

We compare our addition-only UPSI for extended functions (PSI-Cardinality, PSI-Sum, and Circuit-PSI) with [RR22] (RR22) and [CGS22] (CGS22) in Table 2 with total set sizes ranging from 2^{18} to 2^{22} and update sizes from 2^6 to 2^{10}. Our computation and communication complexity grows logarithmically with the total set size and linearly with the update size N_d, so our protocols are more competitive in larger input sizes and smaller update sizes. Note that [CGS22] (CGS22) presents two constructions ($\textsf{C-PSI}_1$ and $\textsf{C-PSI}_2$) with different trade offs between computation and communication, but for all the parameters we choose, $\textsf{C-PSI}_2$ outperforms $\textsf{C-PSI}_1$ in all aspects. We were unable to run CGS22 with input size of 2^{22}, so we use the communication cost and running time under LAN reported in their paper [CGS22], and estimate the running time in the WAN settings.

Communication: Since our communication grows linearly with N_d and only logarithmically with N, our protocols have a communication advantage in settings where $N_d \ll N$. For $N = 2^{18}$, our communication has an improvement of $2.2 - 13\times$ when $N_d = 2^6$ in all functionalities, and when $N_d = 2^8$, $\Pi_{\textsf{UPSI-Add}_{\textsf{ca}}}$ and $\Pi_{\textsf{UPSI-Add}_{\textsf{sum}}}$ have an advantage $1.8 - 3.4\times$. For $N = 2^{20}$, our protocols outperform RR22 by $2.2 - 50\times$ depending on the functionality and update size, with only $\Pi_{\textsf{UPSI-Add}_{\textsf{circuit}}}$ at $N_d = 2^{10}$ not showing an improvement. When $N = 2^{22}$, that improvement extends to all settings and increases to a factor of $2.2 - 200\times$.

Computation: Our computational complexity also grows linearly with N_d and logarithmically with N. Despite this, our computation times do not reflect this asymptotic improvement as clearly, which stems from our usage of costly public key operations. As a result, we show better performance only when N is sufficiently large. In the LAN setting with $N = 2^{20}, N_d = 2^6$, our $\Pi_{\textsf{UPSI-Add}_{\textsf{ca}}}$ and $\Pi_{\textsf{UPSI-Add}_{\textsf{sum}}}$ are faster by $3.2\times$ and $2.1\times$, respectively. By $N = 2^{22}, N_d = 2^6 - 2^8$, our $\Pi_{\textsf{UPSI-Add}_{\textsf{ca}}}, \Pi_{\textsf{UPSI-Add}_{\textsf{sum}}}$ protocols outperform CGS22 by $1.4 - 15\times$.

End to End: Given these communication and computation trade offs, our protocols perform best with more realistic network configurations with lower network bandwidth. At $N = 2^{18}$, we begin to have competitive runtimes for $\Pi_{\textsf{UPSI-Add}_{\textsf{ca}}}$ and $\Pi_{\textsf{UPSI-Add}_{\textsf{sum}}}$ in the smaller update size $N_d = 2^6$. By

N	N_d	Protocol	Comm. (MB)	Total Running Time (s)			
				LAN	200Mbps	50Mbps	5Mbps
2^{18}	–	RR22	37.1	7.76	10.7	13.8	64.4
		CGS22 (C-PSI$_1$)	548	7.90	36.9	106	968
		CGS22 (C-PSI$_2$)	353	6.32	29.4	70.6	619
	2^6	$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{ca}}}$	2.83	7.12	7.59	7.87	11.8
	2^8		11.0	27.6	28.6	30.2	45.6
	2^{10}		42.6	108	110	115	177
	2^6	$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{sum}}}$	5.35	11.0	11.8	12.5	20.1
	2^8		22.3	45.9	47.2	49.3	82.0
	2^{10}		87.1	178	184	195	321
	2^6	$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{circuit}}}$	17.1	81.7	83.1	85.3	110
	2^8		67.0	318	327	330	427
	2^{10}		248	1171	1182	1214	1570
2^{20}	–	RR22	149	31.1	38.4	51.9	258
		CGS22 (C-PSI$_1$)	2190	31.0	135	414	3771
		CGS22 (C-PSI$_2$)	1408	24.3	92.8	268	3872
	2^6	$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{ca}}}$	3.03	7.59	8.14	8.46	12.6
	2^8		11.8	29.6	30.6	32.0	48.7
	2^{10}		45.7	116	121	127	194
	2^6	$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{sum}}}$	5.70	11.8	12.5	13.1	21.5
	2^8		22.3	45.9	47.2	49.3	82.0
	2^{10}		87.1	178	184	195	321
	2^6	$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{circuit}}}$	17.1	81.7	83.1	85.3	110
	2^8		67.0	318	327	330	427
	2^{10}		264	1251	1263	1295	1674
2^{22}	–	RR22	606	125	159	214	1086
		CGS22 (C-PSI$_1$)	6667*	93.0*	126*	226*	1426*
		CGS22 (C-PSI$_2$)	4435*	77.9*	100*	167*	965*
	2^6	$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{ca}}}$	3.22	8.09	9.02	9.33	14.3
	2^8		12.6	31.6	32.7	34.2	52.7
	2^{10}		48.9	123	127	133	205
	2^6	$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{sum}}}$	6.04	12.5	13.3	14.1	23.6
	2^8		23.6	48.8	50.3	53.3	88.6
	2^{10}		92.7	191	197	209	342
	2^6	$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{circuit}}}$	18.1	86.6	88.4	90.2	116
	2^8		71.1	339	343	352	454
	2^{10}		280	1348	1341	1376	1780

Table 2: Communication cost (in MB) and running time (in seconds) comparing our addition-only UPSI protocols to prior work. * indicates estimated communication and running time.

$N = 2^{22}$ and $N_d = 2^6$, our protocols outperform in all network settings by $15 - 76\times$ for $\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{ca}}}$, $11 - 46\times$ for $\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{sum}}}$, and $1.8 - 9.4\times$ for $\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{circuit}}}$.

6.3 UPSI-Cardinality/Sum with Addition and Deletion

Our performance for $\Pi_{\mathsf{UPSI\text{-}Del}_{\mathsf{ca}}}$ and $\Pi_{\mathsf{UPSI\text{-}Del}_{\mathsf{sum}}}$ in comparison with [RR22, CGS22] is presented in Table 3. Since the two protocols are implemented in the same way except that P_0's inputting payloads are different, they have close experimental results. We combine them in the table. This protocol is more expensive than the addition-only ones, so we set smaller update sizes of $N_d = 2^4, 2^5, 2^6$ to demonstrate the turning point where our protocols start to perform better. Our experiments for input size $N = 2^{22}$ are run on a Google Cloud `c2-standard-30` virtual machine with 120 GB of RAM as we run out of 64 GB memory.

N	N_d	Protocol	Comm. (MB)	Total Running Time (s)			
				LAN	200Mbps	50Mbps	5Mbps
2^{20}	–	RR22	149	31.1	38.4	51.9	258
		CGS22 (C-PSI$_1$)	2190	31.0	135	414	3771
		CGS22 (C-PSI$_2$)	1408	24.3	92.8	415	3872
	2^4	$\Pi_{\text{UPSI-Del}_{\text{ca}}}$ $\Pi_{\text{UPSI-Del}_{\text{sum}}}$	58.5	96.1	101	106	179
	2^5		116	190	198	212	362
	2^6		231	364	375	402	723
2^{22}	–	RR22	606	125	159	214	1086
		CGS22 (C-PSI$_1$)	6667*	93.0*	126*	226*	1426*
		CGS22 (C-PSI$_2$)	4435*	77.9*	100*	167*	965*
	2^4	$\Pi_{\text{UPSI-Del}_{\text{ca}}}$ $\Pi_{\text{UPSI-Del}_{\text{sum}}}$	61.4	103	107	113	191
	2^5		122	203	210	223	383
	2^6		243	385	399	429	765

Table 3: Communication cost (in MB) and running time (in seconds) of our protocols for UPSI-Cardinality and UPSI-Sum with addition and deletion in comparison with prior work. * indicates estimated communication and running time.

Communication: Our communication complexity is $O(N_d \cdot \log N)$, but the improvements are not as stark, for two reasons: (1) the increased stash and node sizes required, and (2) in addition to exchanging ciphertexts, the parties also perform OT and garbled circuits. Despite this, our protocol still achieves lower communication overhead in most settings. At $N = 2^{20}$, our communication has an improvement of $1.3 - 2.5\times$ when $N_d \leq 2^5$. By $N = 2^{22}$, our communication has an improvement of $2.5 - 9.9\times$ for all update sizes.

Computation: Our performance under LAN is again dominated by public key operations, but, unlike in the addition-only protocols, does not benefit from the efficient El Gamal instantiations. Our computation has the same growth rate as communication, and so we expect our performance to eventually beat CGS22 when N is sufficiently large.

End to End: As shown in Table 3, the end to end running time of our protocol begins to outperform RR22 and CGS22 at 5 Mbps when $N = 2^{20}, N_d = 2^4$ by $1.4\times$. By $N = 2^{22}$, we show an improvement of $1.3 - 5.1\times$ at 5 Mbps for all update sizes, and an improvement of $1.5\times$ at 50 Mbps for $N_d = 2^4$.

6.4 UPSI for Plain PSI

We compare our plain UPSI protocols with [RR22] (RR22) in Table 4. We evaluate two constructions in RR22 with different encoding sizes of $1.28n$ and $1.23n$, which have different trade offs in computation and communication, denoted as `fast` and `small` respectively in the table. Note that our addition-only plain UPSI (Figure 5) contains only one encrypted tree, hence it is more efficient than our other addition-only protocols. To best demonstrate our turning point, we use $N_d = 2^4, 2^6, 2^8, 2^{10}$ for $\Pi_{\text{UPSI-Add}_{\text{psi}}}$ and $N_d = 2^4, 2^5, 2^6$ for $\Pi_{\text{UPSI-Del}_{\text{psi}}}$

Communication: Similarly as in our other protocols, our communication complexity in both $\Pi_{\text{UPSI-Add}_{\text{psi}}}$ and $\Pi_{\text{UPSI-Del}_{\text{psi}}}$ are $O(N_d \cdot \log N)$. The communication cost of $\Pi_{\text{UPSI-Add}_{\text{psi}}}$ outperforms

N	N_d	Protocol	Comm. (MB)	Total Running Time (s)			
				LAN	200Mbps	50Mbps	5Mbps
2^{20}	–	RR22 (fast)	34.3	0.73	3.09	7.10	55.9
		RR22 (small)	32.1	1.00	3.21	6.97	52.8
	2^4	$\Pi_{\mathsf{UPSI\text{-}Add_{psi}}}$	0.50	1.41	1.84	1.89	2.48
	2^6		1.95	5.54	6.11	6.30	8.88
	2^8		7.57	21.6	22.8	23.5	34.1
	2^{10}		29.6	84.9	87.5	90.8	133
	2^4	$\Pi_{\mathsf{UPSI\text{-}Del_{psi}}}$	58.7	98.6	103	109	181
	2^5		117	195	203	215	369
	2^6		231	370	384	410	729
2^{22}	–	RR22 (fast)	138	3.45	11.3	27.7	227
		RR2 (small)	129	4.81	12.2	27.6	214
	2^4	$\Pi_{\mathsf{UPSI\text{-}Add_{psi}}}$	0.53	1.49	1.93	1.97	2.57
	2^6		2.06	5.89	6.48	6.67	9.51
	2^8		8.03	22.9	24.1	24.9	36.2
	2^{10}		31.5	89.9	92.8	96.2	141
	2^4	$\Pi_{\mathsf{UPSI\text{-}Del_{psi}}}$	61.6	105	109	115	194
	2^5		122	208	214	228	388
	2^6		243	396	412	437	776

Table 4: Communication cost (in MB) and running time (in seconds) of our protocols for plain UPSI in comparison with prior work.

RR22 by $1.1 - 240\times$ in all settings, whereas that of $\Pi_{\mathsf{UPSI\text{-}Del_{psi}}}$ only beats RR22 by $1.1 - 2.1\times$ with $N = 2^{22}$ and $N_d = 2^4, 2^5$.

Computation: Our computation complexity is similar to communication, leading to better performance when N is sufficiently large. Our addition-only protocol starts to outperforms RR22 when $N = 2^{22}$ and $N_d = 2^4$.

End to End: As the communication and computation discussed above, our protocols are more competitive with larger input sizes, smaller updates, and in networks with lower bandwidths. By $N = 2^{22}$ and $N_d = 2^4$, $\Pi_{\mathsf{UPSI\text{-}Add_{psi}}}$ achieves an improvement of $2.3 - 88\times$ in all network settings. It outperforms RR22 by $1.5\times$ even when the update size grows to 2^{10}.

6.5 Worst-Case Logarithmic Complexity

We compare our one-sided addition-only plain UPSI protocol $\Pi_{\mathsf{UPSI\text{-}Add_{psi}}}$ with that of [BMX22] (BMX22). While BMX22 has amortized complexity of $O(N_d \cdot \log N)$, their worst-case complexity is $O(N)$ when they update the leaf level of the tree. By plugging in our new oblivious data structure, we significantly reduce the worst-case complexity to $O(N_d \cdot \log N)$. The worst-case performance (Max) and amortized performance (Avg) are presented in Table 5 with $N = 2^{18}, 2^{20}$ and $N_d = 2^6, 2^8, 2^{10}$. To analyze the amortized cost of BMX22, we start with two sets each of size N. Then, on every new day, both parties add a new set of size N_d to their existing sets and run the UPSI protocol. We repeat this process over a period of several days ($\frac{N}{N_d}$) until the total set size of each party reaches $2N$. We report the amortized cost over these $\frac{N}{N_d}$ days.

N	N_d	Protocol	Comm.(MB)		Total Running Time(s)							
			Max	Avg	LAN		200Mbps		50Mbps		5Mbps	
					Max	Avg	Max	Avg	Max	Avg	Max	Avg
2^{18}	2^6	BMX22	120	1.09	79.6	4.30	85.9	4.53	100	4.59	272	5.88
		$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{psi}}}$	1.82		5.17		6.24		6.31		8.70	
	2^8	BMX22	121	3.74	77.9	14.7	84.2	15.1	98.3	15.5	268	20.3
		$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{psi}}}$	7.08		20.2		21.8		22.6		32.4	
	2^{10}	BMX22	122	12.5	86.4	49.0	87.7	49.9	95.1	51.3	268	67.2
		$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{psi}}}$	27.7		79.4		81.5		84.7		124	
2^{20}	2^6	BMX22	480	1.25	321	4.92	350	5.17	403	5.24	1090	6.76
		$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{psi}}}$	1.95		5.54		6.11		6.30		8.88	
	2^8	BMX22	481	4.37	319	17.2	344	17.6	401	18.1	1090	23.7
		$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{psi}}}$	7.57		21.6		22.8		23.5		34.1	
	2^{10}	BMX22	482	15.0	312	58.9	337	59.9	394	61.4	1090	81.1
		$\Pi_{\mathsf{UPSI\text{-}Add}_{\mathsf{psi}}}$	29.6		84.9		87.5		90.8		133	

Table 5: Communication cost (in MB) and running time (in seconds) comparing our addition-only plain UPSI protocol to the worst-case and average-case performance of [BMX22].

Comparison. As shown in Table 5, our communication cost is comparable to BMX22's average-case while outperforming their worst-case by $4.4-246\times$ in all settings since their worst-case communication grows linearly with N. Similarly, our computation cost is comparable to their average-case while outperforming their worst-case by $1.1-58\times$ in the LAN setting. As a result, the end to end running time of our protocol outperforms BMX22's worst-case in all settings by $1.1-123\times$, while having $1.1-1.8\times$ overhead compared to their average-case. Concerning the worst-case performance, our protocol has more advantages in larger input sizes and smaller updates.

Acknowledgments

This project is supported in part by the NSF CNS Award 2247352, Brown Data Science Seed Grant, Meta Research Award, Google Research Scholar Award, and Amazon Research Award.

References

[ACG+24] Archita Agarwal, David Cash, Marilyn George, Seny Kamara, Tarik Moataz, and Jaspal Singh. Updatable private set intersection from structured encryption. Cryptology ePrint Archive, 2024. `https://eprint.iacr.org/2024/1183`.

[ADMT22] Aydin Abadi, Changyu Dong, Steven J. Murdoch, and Sotirios Terzis. Multi-party updatable delegated private set intersection. In Ittay Eyal and Juan A. Garay, editors, *FC 2022*, volume 13411 of *LNCS*, pages 100–119. Springer, Cham, May 2022.

[Ads] Private Intersection-Sum Protocols with Applications to Attributing Aggregate Ad Conversions. `https://research.google/pubs/pub51026/`.

[ADT11] Giuseppe Ateniese, Emiliano De Cristofaro, and Gene Tsudik. (If) size matters: Size-hiding private set intersection. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 156–173. Springer, Berlin, Heidelberg, March 2011.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 535–548. ACM Press, November 2013.

[App] Privacy-Preserving Contact Tracing. https://covid19.apple.com/contacttracing.

[BBV+20] Alex Berke, Michiel A. Bakker, Praneeth Vepakomma, Ramesh Raskar, Kent Larson, and Alex 'Sandy' Pentland. Assessing disease exposure risk with location histories and protecting privacy: A cryptographic approach in response to A global pandemic. *CoRR*, abs/2003.14412, 2020.

[BCG+22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 603–633. Springer, Cham, August 2022.

[BKC+23] Saikrishna Badrinarayanan, Ranjit Kumaresan, Mihai Christodorescu, Vinjith Nagaraja, Karan Patel, Srinivasan Raghuraman, Peter Rindal, Wei Sun, and Minghua Xu. A plug-n-play framework for scaling private set intersection to billion-sized sets. In *Cryptology and Network Security - 22nd International Conference, CANS 2023, Augusta, GA, USA, October 31 - November 2, 2023, Proceedings*, volume 14342 of *Lecture Notes in Computer Science*, pages 443–467. Springer, 2023.

[BMX22] Saikrishna Badrinarayanan, Peihan Miao, and Tiancheng Xie. Updatable private set intersection. *PoPETs*, 2022(2):378–406, April 2022.

[BPSY23] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Near-optimal oblivious key-value stores for efficient psi, PSU and volume-hiding multi-maps. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. USENIX Association, 2023.

[CGS22] Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-PSI with linear complexity via relaxed batch OPPRF. *PoPETs*, 2022(1):353–372, January 2022.

[CHLR18] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1223–1237. ACM Press, October 2018.

[Chr] Protect your accounts from data breaches with Password Checkup. https://security.googleblog.com/2019/02/protect-your-accounts-from-data.html.

[CILO22] Wutichai Chongchitmate, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. PSI from ring-OLE. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 531–545. ACM Press, November 2022.

[clo] Google Cloud. https://cloud.google.com.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1243–1255. ACM Press, October / November 2017.

[CM20] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 34–63. Springer, Cham, August 2020.

[CMdG+21] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1135–1150. ACM Press, November 2021.

[CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 502–534, Virtual Event, August 2021. Springer, Cham.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 789–800. ACM Press, November 2013.

[DT10] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In Radu Sion, editor, *FC 2010*, volume 6052 of *LNCS*, pages 143–159. Springer, Berlin, Heidelberg, January 2010.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[GMR+21] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private set operations from oblivious switching. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 591–617. Springer, Cham, May 2021.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GPR+21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 395–425, Virtual Event, August 2021. Springer, Cham.

[gRP] Google Remote Procedure Call (gPRC). `https://grpc.io`.

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

[HFH99] Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 78–86. ACM, 1999.

[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 *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 370–389. IEEE, 2020.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Berlin, Heidelberg, August 2003.

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

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 818–829. ACM Press, October 2016.

[KMRS14] Seny Kamara, Payman Mohassel, Mariana Raykova, and Saeed Sadeghian. Scaling private set intersection to billion-element sets. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security*, pages 195–215, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[KRS+19] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019*, pages 1447–1464. USENIX Association, August 2019.

[Mea86] Catherine Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 7-9, 1986*, pages 134–137. IEEE Computer Society, 1986.

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

[MPR+20] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 3–33. Springer, Cham, August 2020.

[OOS17] Michele Orrù, Emmanuela Orsini, and Peter Scholl. Actively secure 1-out-of-N OT extension with application to private set intersection. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 381–396. Springer, Cham, February 2017.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[PJC] Private Join and Compute. `https://github.com/google/private-join-and-compute`.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 401–431. Springer, Cham, August 2019.

[PRTY20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 739–767. Springer, Cham, May 2020.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX Security 2015*, pages 515–530. USENIX Association, August 2015.

[PSTY19] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 122–153. Springer, Cham, May 2019.

[PSWW18] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 125–157. Springer, Cham, April / May 2018.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 797–812. USENIX Association, August 2014.

[RR] Srinivasan Raghuraman and Peter Rindal. VOLE-PSI. `https://github.com/Visa-Research/volepsi`.

[RR17] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 235–259. Springer, Cham, April / May 2017.

[RR22] Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2505–2517. ACM Press, November 2022.

[RRT23] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 602–632. Springer, Cham, August 2023.

[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 901–930. Springer, Cham, October 2021.

[Sig] Technology preview: Private contact discovery for Signal. `https://signal.org/blog/private-contact-discovery/`.

[SvS+13] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 299–310. ACM Press, November 2013.

[TSS+20] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight contact tracing with strong privacy. *IEEE Data Eng. Bull.*, 43(2):95–107, 2020.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient Multi-Party computation toolkit. `https://github.com/emp-toolkit`, 2016.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Addition-Only Plain UPSI

One-sided addition-only plain UPSI with worst-case complexity can be achieved with slight modifications of the protocol in [BMX22] and we present a protocol $\Pi_{\mathsf{UPSI\text{-}Add_{psi}}}$ in Figure 11. By plugging in our new oblivious data structure, the worst-case communication and computation can be reduced to $O(N_d \cdot \log N)$ from $O(N)$. Benchmarks for this protocol can be found in Table 4.

B Proof of Theorem 3.2

Correctness. We first show correctness of our protocols by induction. On day 0, all sets are initialized as empty sets so correctness trivially holds. Next, on any day d, our goal is to compute $I_d = (X \cup X_d) \cap (Y \cup Y_d)$. First, from the correctness of the subroutines $\mathsf{GetPath}$ and $\mathsf{UpdateTree}$ as well as the threshold additive homomorphic encryption scheme, one can observe that in the output

Initialization:

1. P_0 and P_1 independently generate key pairs for an additive homomorphic encryption scheme $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and share the public keys. P_0 and P_1 sample $\mathsf{k}_0, \mathsf{k}_1 \xleftarrow{\$} \mathbb{Z}_q$, respectively. Both parties agree on a randomly sampled PRF key $\mathsf{k} \xleftarrow{\$} \{0,1\}^\lambda$.
2. P_0 and P_1 generate initial trees with only an empty root and stash: $(\widetilde{\mathcal{D}}, \widetilde{\mathcal{S}})$ and $(\mathcal{D}, \mathcal{S})$, respectively. Initialize $I_0 = \emptyset$.

Day *d*: P_0 and P_1 hold $(\widetilde{\mathcal{D}}, \widetilde{\mathcal{S}})$ and $(\mathcal{D}, \mathcal{S})$, respectively. Let L be the tree height of $\mathcal{D}$ and $\widetilde{\mathcal{D}}$. Both parties update L as they update the trees below. P_0 also holds H_X. Let X, Y denote the two parties' sets at the end of the previous day, respectively. P_0 holds a new input set X_d and P_1 holds a new input set Y_d. Let $n = |X_d|$ and $m = |Y_d|$.

1. **P_0 learns $X \cap Y_d$.**
   (a) P_1 computes $H(Y_d)^{\mathsf{k}_1}$ and sends to P_0.
   (b) P_0 raises each element by k_0 to obtain $H(Y_d)^{\mathsf{k}_0\mathsf{k}_1}$ and compares against H_X (which equals $H(X \setminus I_{d-1})^{\mathsf{k}_0\mathsf{k}_1}$) to learn $I_Y = X \cap Y_d$.
2. **Y tree update.** P_1 sends $(\{(\widetilde{\mathsf{updates}}_i, \ell_i)\}_{i=1}^m, \widetilde{\mathcal{S}}'_1) \leftarrow \mathsf{UpdateTree}(Y, \perp, \mathcal{D}, \mathcal{S}, F_\mathsf{k}(\cdot), \mathsf{Enc}_{\mathsf{pk}_1}(\cdot))$ to P_0. P_0 replaces each path $\mathcal{P}(\ell_j)$ with $\widetilde{\mathsf{updates}}_j$ in $\widetilde{\mathcal{D}}$, and replaces $\widetilde{\mathcal{S}}$ with $\widetilde{\mathcal{S}}'$.
3. **P_0 learns $X_d \cap (Y \cup Y_d)$.** P_0 samples a random mask $\alpha \xleftarrow{\$} \mathbb{Z}_q$ and sends $\mathsf{Enc}_{\mathsf{pk}_0}(\alpha)$ to P_1, then for every element $x_i \in X_d$:
   (a) P_0 obtains $\{\mathsf{Enc}_{\mathsf{pk}_1}(y_{i,j})\}_{j=1}^{\sigma \cdot L+\rho} \leftarrow \mathsf{GetPath}(\widetilde{\mathcal{D}}, \widetilde{\mathcal{S}}, F_\mathsf{k}(\cdot), x_i)$, samples a random mask β_i, and homomorphically computes $\widetilde{\mathsf{path}} = \{\beta_i \odot (\mathsf{Enc}_{\mathsf{pk}_1}(y_{i,j}) \ominus \mathsf{Enc}_{\mathsf{pk}_1}(x_i)) \oplus \mathsf{Enc}_{\mathsf{pk}_1}(\alpha)\}_{j=1}^{\sigma \cdot L+\rho}$ and sends $\widetilde{\mathsf{path}}$ to P_1.
   (b) For each $\mathsf{ct}_j \in \widetilde{\mathsf{path}}$, P_1 decrypts the ciphertext to obtain $\gamma_j = \mathsf{Dec}_{\mathsf{sk}_1}(\mathsf{ct}_j)$, samples a random mask $\delta_j \xleftarrow{\$} \mathbb{Z}_q$, and homomorphically computes $\widetilde{\mathsf{res}}_j = \delta_j \cdot (\mathsf{Enc}_{\mathsf{pk}_0}(\gamma_j) \ominus \mathsf{Enc}_{\mathsf{pk}_0}(\alpha)$.
   (c) P_1 samples a random permutation π from $[\sigma \cdot L + \rho]$ and sends $\pi(\{\widetilde{\mathsf{res}}_j\}_{j=1}^{\sigma \cdot L+\rho})$ to P_0.
   (d) P_0 adds x_i to I_X if $\mathsf{Dec}_{\mathsf{sk}_0}(\widetilde{\mathsf{res}}_j) = 0$ for any $\widetilde{\mathsf{res}}_j$.
4. **Output Generation.** P_0 outputs $I_d = I_Y \cup I_X$.
5. **H_X updates.**
   (a) P_0 creates a set $X'_d = (X_d \setminus I_X)$ and then pads it with dummy elements until $|X'_d| = |X_d|$. He then samples $\varepsilon \leftarrow \mathbb{Z}_q$ and sends $H(X'_d)^{\varepsilon \mathsf{k}_0}$ to P_1.
   (b) P_1 raises each element in $H(X'_d)^{\varepsilon \mathsf{k}_0}$ to the power k_1 to obtain $H(X'_d)^{\varepsilon \mathsf{k}_0 \mathsf{k}_1}$ which she sends back to P_0.
   (c) P_0 raises each element in $H(X'_d)^{\varepsilon \mathsf{k}_0 \mathsf{k}_1}$ to the power of ε^{-1} to obtain $H(X'_d)^{\mathsf{k}_0 \mathsf{k}_1}$ from which he derives $H(X_d \setminus I_X)^{\mathsf{k}_0\mathsf{k}_1}$.
   (d) P_0 updates $H_X = (H_X \setminus H(I_Y)^{\mathsf{k}_0\mathsf{k}_1}) \cup H(X_d \setminus I_X)^{\mathsf{k}_0\mathsf{k}_1}$.

Figure 11: Protocol $\Pi_{\mathsf{UPSI\text{-}Add_{psi}}}$ for addition-only plain UPSI $\mathcal{F}_{\mathsf{UPSI\text{-}Add_{psi}}}$.

generation, in Step 6, the output computed is indeed using the candidates from Step 2 and Step 3. In particular, we can expand I_d as

$$\begin{aligned} I_d &= (X \cup X_d) \cap (Y \cup Y_d) \\ &= ((X \cup X_d) \cap Y) \cup ((X \cup X_d) \cap Y_d) \\ &= (X \cap Y) \cup (X_d \cap Y) \cup ((X \cup X_d) \cap Y_d) \end{aligned}$$

Recall that $X_d \cap X = \emptyset$ and $Y_d \cap Y = \emptyset$. Hence the new elements to be added to the intersection consists of two disjoint sets, $(X_d \cap Y)$ obtained in Step 2 and $((X \cup X_d) \cap Y_d)$ obtained in Step 3.

Imported Lemma B.1 (Theorem 1 from [SvS+13]). *Let $\{x_i\}_{i=1}^N$ be any sequence of elements being added to a binary tree via* $\mathsf{UpdateTree}$*. Let tree node size $\sigma = 5$, tree height $L = \lceil \log N \rceil$, and stash size ρ. If the function $F(\cdot)$ is a random function, namely $F(x_i)$ outputs a random leaf node, then the probability of failure (abort in Step 2d) after a sequence of* $\mathsf{UpdateTree}$ *operations corresponding to $\{x_i\}_{i=1}^N$ can be bounded by* $\Pr[|\mathcal{S}| > \rho] < 14 \cdot (0.6002)^\rho$.

Security Against Corrupted P_0. The simulator Sim_0 can be constructed that simulates P_0's view as follows. On input $(1^\lambda, X_{[D]}, \mathcal{F}_0(X_{[D]}, Y_{[D]}))$, 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 5, Sim_0 will change m_3 depending on the functionality. Let Γ_d be the number of elements in m_3 for an honest P_1 on day d, which is derivable from the public parameters.
  - In $\mathcal{F}_{\mathsf{UPSI\text{-}Add_{ca}}}$, $\mathcal{F}_0(X_{[D]}, Y_{[D]}) = \{C_1, C_2, \ldots, C_D\}$ where C_d is the number of elements in the intersection after day d. Let $C := C_d - C_{d-1}$. Sim_0 samples $\alpha_i \xleftarrow{\$} \mathbb{Z}_q$ for all $i \in [\Gamma_d - C]$ and a random permutation π over $[\Gamma_d]$, and sends the message:
  $$m_3 = \pi\left(\{\mathsf{PartDec}_{\mathsf{sk}_1}(\mathsf{Enc}_{\mathsf{pk}}(\alpha_i))\}_{i=1}^{\Gamma_d - C} \cup \{\mathsf{PartDec}_{\mathsf{sk}_1}(\mathsf{Enc}_{\mathsf{pk}}(0))\}_{j=1}^{C}\right).$$
  - In $\mathcal{F}_{\mathsf{UPSI\text{-}Add_{sum}}}$, $\mathcal{F}_0(X_{[D]}, Y_{[D]}) = \{(C_1, V_1), (C_2, V_2), \ldots, (C_D, V_D)\}$ where C_d is the number of elements in the intersection and V_d is the sum of associated values in the intersection after day d. Let $C := C_d - C_{d-1}$. Sim_0 samples $\alpha_i, \beta_i \xleftarrow{\$} \mathbb{Z}_q$ for all $i \in [\Gamma_d - C]$ and $\gamma_j \xleftarrow{\$} \mathbb{Z}_q$ for all $j \in [C]$. It also sample a random permutation π over $[\Gamma_d]$ and sends the message:
  $$m_3 = \pi\left(\left\{\begin{pmatrix} \mathsf{PartDec}_{\mathsf{sk}_1}(\mathsf{Enc}_{\mathsf{pk}}(\alpha_i)), \\ \mathsf{Enc}_{\mathsf{pk}}(\beta_i) \end{pmatrix}\right\}_{i=1}^{\Gamma_d - C} \bigcup \left\{\begin{pmatrix} \mathsf{PartDec}_{\mathsf{sk}_1}(\mathsf{Enc}_{\mathsf{pk}}(0)), \\ \mathsf{Enc}_{\mathsf{pk}}(\gamma_j) \end{pmatrix}\right\}_{j=1}^{C}\right).$$
  - In $\mathcal{F}_{\mathsf{UPSI\text{-}Add_{circuit}}}$, $\mathcal{F}_0(X_{[D]}, Y_{[D]}) = \{(C_1, Z_1), (C_2, Z_2), \ldots, (C_D, Z_D)\}$ where C_d is the number of elements in the intersection, $Z_d = \{[\![z_1]\!]_0, [\![z_2]\!]_0, \ldots, [\![z_{C_d - C_{d-1}}]\!]_0\}$ is a set of P_0's secret shares for elements added to the intersection on day d. Let $C := C_d - C_{d-1}$. Sim_0 samples $\alpha_i, \beta_i \xleftarrow{\$} \mathbb{Z}_q$ for all $i \in [\Gamma_d - C]$ and $\gamma_j \xleftarrow{\$} \mathbb{Z}_q$ for all $j \in [C]$. It also samples a random permutation π over $[\Gamma_d]$ and sends the message:
  $$m_3 = \pi\left(\left\{\begin{pmatrix} \mathsf{PartDec}_{\mathsf{sk}_1}(\mathsf{Enc}_{\mathsf{pk}}(\alpha_i)), \\ \mathsf{Enc}_{\mathsf{pk}}(\beta_i) \end{pmatrix}\right\}_{i=1}^{\Gamma_d - C} \bigcup \left\{\begin{pmatrix} \mathsf{PartDec}_{\mathsf{sk}_1}(\mathsf{Enc}_{\mathsf{pk}}(0)), \\ \mathsf{Enc}_{\mathsf{pk}}(\gamma_j) \end{pmatrix}\right\}_{j=1}^{C}\right).$$
- In Step 6, an honest P_1 sends a message m_4' in $\Pi_{\mathsf{UPSI\text{-}Add_{sum}}}$, where Sim_0 will instead send $m_4' = \mathsf{PartDec}_{\mathsf{sk}_1}(\mathsf{Enc}_{\mathsf{pk}}(V))$ where $V := V_d - V_{d-1}$ on day d.
- In Step 7, Sim_0 samples a random set $Y_d' \xleftarrow{\$} \mathbb{Z}_q^{|Y_d|}$ and sends $m_5 = (\{(\widetilde{\mathsf{updates}}_j, \ell_j)\}_{j=1}^m, \widetilde{\mathcal{S}}_1) \leftarrow \mathsf{UpdateTree}(Y_d', \perp, \mathcal{D}_1, \mathcal{S}_1, F_{\mathsf{k}}(\cdot), \mathsf{Enc}_{\mathsf{pk}}(\cdot))$.

Finally, Sim_0 outputs P_0's view. Using a hybrid argument, we can prove for any $D \in \mathbb{N}$ and any inputs $(X_{[D]}, Y_{[D]})$,

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

Hyb_0: P_0's view together with P_1's output in the real protocol.

Hyb_1: Same as Hyb_0 except that P_1's output is replaced with $\mathcal{F}_1(X_{[D]}, Y_{[D]})$. This follows from the correctness of the protocol.

Hyb_2: Same as Hyb_1 except that in $\Pi_{\mathsf{UPSI\text{-}Add_{sum}}}$, m'_4 is replaced with $\mathsf{PartDec}_{\mathsf{sk}_1}(\mathsf{Enc}_{\mathsf{pk}}(V))$ for $V = V_d - V_{d-1}$ on each day $d \in [D]$. This hybrid is statistically indistinguishable from Hyb_1 by the re-randomization of the encryption scheme.

$\mathsf{Hyb}_{3,i}$: Same as Hyb_2 except the first element in each tuple of m_3 is replaced with a partial decryption of a fresh encryption of $\alpha_k \cdot (a_k - b_k)$. This is actually a series of hybrids where the element in the i-th tuple is replaced in $\mathsf{Hyb}_{3,i}$. Each hybrid is statistically indistinguishable by the re-randomization of the encryption scheme. Let Hyb_3 be the last hybrid in this series.

$\mathsf{Hyb}_{4,i}$: Same as Hyb_3 except that the first element in each tuple of m_3 is either replaced with partial decryption of a fresh encryption of 0 if $a_k - b_k = 0$ or of a uniformly random element in $\mathbb{Z}_q$ otherwise. In the first case, $\mathsf{Hyb}_{4,i}$ is statistically indistinguishable from $\mathsf{Hyb}_{4,i-1}$ by the re-randomization of the encryption scheme. In the second case, $\mathsf{Hyb}_{4,i}$ is statistically indistinguishable from $\mathsf{Hyb}_{4,i-1}$ because $\alpha_k \cdot (a_k - b_k)$ has a uniform distribution over $\mathbb{Z}_q$ when $\alpha_k \xleftarrow{\$} \mathbb{Z}_q$ and $a_k - b_k \neq 0$. This is also a series of hybrids where the element in the i-th tuple is replaced in $\mathsf{Hyb}_{4,i}$. Let Hyb_4 be the last hybrid in this series.

$\mathsf{Hyb}_{5,i}$: Same as Hyb_4 except that the second element in each tuple of m_3 is replaced depending on the functionality, where the i-th tuple has its element replaced in $\mathsf{Hyb}_{5,i}$.

In $\Pi_{\mathsf{UPSI\text{-}Add_{sum}}}$ and $\Pi_{\mathsf{UPSI\text{-}Add_{circuit}}}$, it is replaced with a fresh encryption of a random value. This hybrid is computationally indistinguishable by the CPA security of the encryption scheme.

In $\Pi_{\mathsf{UPSI\text{-}Add_{ca}}}$, nothing changes in these hybrids.

Let Hyb_5 be the last hybrid in this series of hybrids.

Hyb_6: Same as Hyb_5 except that P_1 never aborts in Step 2d of $\mathsf{UpdateTree}$. This hybrid computationally indistinguishable from Hyb_5 because of the pseudorandomness of $F_{\mathsf{k}}(\cdot)$. If one can distinguish between Hyb_5 and Hyb_6, then it means the abort probability in Hyb_5 is non-negligible. By Imported Lemma B.1, if the function $F(\cdot)$ used in $\mathsf{UpdateTree}$ is a random function, then the probability of abort is negligible. Hence we can use the abort probability to distinguish between a pseudorandom function $F_{\mathsf{k}}(\cdot)$ and a random function $F(\cdot)$.

Hyb_7: Same as Hyb_6 except that Y_d is replaced with Y'_d in $\mathsf{UpdateTree}$ for m_5. By the construction of $\mathsf{UpdateTree}$ and CPA security of the encryption scheme, this hybrid is computationally indistinguishable. P_0's view in this hybrid is exactly Sim_0's output, concluding the proof.

Security Against Corrupted P_1. Sim_1 can be constructed that simulates P_1's view as follows. On input $(1^\lambda, Y_{[D]}, \mathcal{F}_1(X_{[D]}, Y_{[D]}))$, 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, Sim_1 samples a random set $X'_d \xleftarrow{\$} \mathbb{Z}_q^{|X_d|}$ and random associated values $P'_d \xleftarrow{\$} \mathbb{Z}_q^{|X_d|}$ and sends $m_1 = (\{(\widetilde{\mathsf{updates}}_i, \ell_i)\}_{i=1}^n, \widetilde{\mathcal{S}}_0) \leftarrow \mathsf{UpdateTree}(X'_d, P'_d, \mathcal{D}_0, \mathcal{S}_0, F_{\mathsf{k}}(\cdot), \mathsf{Enc}_{\mathsf{pk}}(\cdot))$.
- In Step 3, Sim_1 will sample random values $\alpha_i, \beta_i \xleftarrow{\$} \mathbb{Z}_q$ for all $i \in [\Lambda_d]$ where $\Lambda_d = |m_2|$ is the number of elements in m_2 for an honest P_0, and sends the message $m_2 = \{(\mathsf{Enc}_{\mathsf{pk}}(\alpha_i), \mathsf{Enc}_{\mathsf{pk}}(\beta_i))\}_{i=1}^{\Lambda_d}$.

- In Step 6, in $\Pi_{\mathsf{UPSI\text{-}Add_{sum}}}$, Sim_1 samples a random value $\gamma \leftarrow \mathbb{Z}_q$ and sends $m_4 = \mathsf{Enc_{pk}}(\gamma)$.

  In $\Pi_{\mathsf{UPSI\text{-}Add_{circuit}}}$, $\mathcal{F}_0(X_{[D]}, Y_{[D]}) = \{(C_1, Z_1), (C_2, Z_2), \ldots, (C_D, Z_D)\}$ where C_d is the number of elements in the intersection, $Z_d = \{[\![z_1]\!]_1, [\![z_2]\!]_1, \ldots, [\![z_{C_d - C_{d-1}}]\!]_1\}$ is a set of P_1's secret shares for elements added to the intersection on day d. Sim_1 sends the message

$$m_4 = \{\mathsf{PartDec_{sk_0}}(\mathsf{Enc_{pk}}([\![z_i]\!]_1))\}_{i=1}^{C_d - C_{d-1}}.$$

Finally, Sim_1 outputs P_1's view. Using a hybrid argument, we can prove for any $D \in \mathbb{N}$ and any inputs $(X_{[D]}, Y_{[D]})$,

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

Hyb_0: P_1's view together with P_0's output in the real protocol.

Hyb_1: Same as Hyb_0 except that P_0's output in $\Pi_{\mathsf{UPSI\text{-}Add_{circuit}}}$ is replaced with $\mathcal{F}_0(X_{[D]}, Y_{[D]})$. This follows from the correctness of the protocol.

$\mathsf{Hyb}_{2,i}$: Same as Hyb_1 except that in $\Pi_{\mathsf{UPSI\text{-}Add_{circuit}}}$, Sim_1 replaces m_4 with a partial decryption of a fresh encryption of $p_k - [\![z_k]\!]_0$. This is actually a series of hybrids where the i-th element is replaced in $\mathsf{Hyb}_{2,i}$. These hybrids are statistically indistinguishable by the re-randomization of the encryption scheme. Note that $p_k - [\![z_k]\!]_0 = [\![z_k]\!]_1$ so this can also be seen as a fresh encryption of $[\![z_k]\!]_1$. Let Hyb_2 be the last hybrid in this series of hybrids.

Hyb_3: Same as Hyb_2 except that in $\Pi_{\mathsf{UPSI\text{-}Add_{sum}}}$, m_4 is replaced by a fresh encryption of $\mathsf{Enc_{pk}}(\sum_{k \in K} p_k)$. This is statistically indistinguishable by the re-randomization of the encryption scheme.

Hyb_4: Same as Hyb_3 except that in $\Pi_{\mathsf{UPSI\text{-}Add_{sum}}}$, Sim_1 samples $\gamma \xleftarrow{\$} \mathbb{Z}_q$ and replaces m_4 with $\mathsf{Enc_{pk}}(\gamma)$. This is computationally indistinguishable by the CPA security of the encryption scheme.

$\mathsf{Hyb}_{5,i,b}$: Same as Hyb_4 except that each tuple in m_2 is replaced with a tuple of fresh encryptions: $(\mathsf{Enc}(y_{i,j} - x_i), \mathsf{Enc_{pk}}(p_i))$. This is a series of hybrids where the b-th element of the i-th tuple is replaced in $\mathsf{Hyb}_{5,i,b}$. These are statistically indistinguishable by the re-randomization of the encryption scheme. Let Hyb_5 be the last hybrid in this series of hybrids.

$\mathsf{Hyb}_{6,i,b}$: Same as Hyb_5 except that each tuple in m_2 is replaced with a tuple of fresh encryptions of random values. This is a series of hybrids where the b-th element of the i-th tuple is replaced in $\mathsf{Hyb}_{6,i,b}$. These are computationally indistinguishable by the CPA security of the encryption scheme. Let Hyb_6 be the last hybrid in this series of hybrids.

Hyb_7: Same as Hyb_6 except that P_0 never aborts in Step 2d of $\mathsf{UpdateTree}$. This hybrid computationally indistinguishable from Hyb_6 because of the pseudorandomness of $F_{\mathsf{k}}(\cdot)$ and Imported Lemma B.1.

Hyb_8: Same as Hyb_7 except that X_d is replaced with X'_d and P_d is replaced with P'_d in $\mathsf{UpdateTree}$ for m_1. By the construction of $\mathsf{UpdateTree}$ and CPA security of the encryption scheme, this hybrid is computationally indistinguishable. P_1's view in this hybrid is exactly Sim_1's output, concluding the proof.

C Proof for Subprotocol $\Pi_{\mathsf{CombinePath}}$

In this section, we prove correctness and security of the subprotocol $\Pi_{\mathsf{CombinePath}}$ presented in Section 4.2.

Correctness. Given the correctness of the additively homomorphic encryption scheme and $\mathcal{F}_{\mathsf{lookup}}$, we prove that $\Pi_{\mathsf{CombinePath}}$ correctly outputs shares $[\![\sum_{x=y_i}(p \cdot q_i)]\!]$ over $\mathbb{Z}_q$ where (x, p) are inputs to the subroutine and (y_i, q_i) are input as encrypted ciphertexts in $\widetilde{\mathsf{path}}$. Consider first i such that $x \neq y_i$. Per the homomorphic operations done in Step 1, $\gamma_i = y_i - x + \alpha_i$ over $\mathbb{Z}_q$ and so $\mathcal{F}_{\mathsf{lookup}}$ will receive $a = y_i - x + \alpha_i$ and $b = \alpha_i$. Because $x \neq y_i$, $\alpha_i \neq y_i - x + \alpha_i$ over $\mathbb{Z}_q$ and so $\mathsf{Responder}$ will receive $m_{i,1}$ from $\mathcal{F}_{\mathsf{lookup}}$. Per Step 4, $[\![r_i]\!]_1 = \mathsf{Dec}_{\mathsf{sk}}(m_{i,1}) = -\beta_i$ and $[\![r_i]\!]_0 = \beta_i$ and so $r_i = [\![r_i]\!]_0 + [\![r_i]\!]_1 = \beta - \beta = 0$. In the case where $x = y_i$, $\alpha_i = y_i - x + \alpha_i$, the $\mathsf{Responder}$ receives $m_{i,0}$, and $[\![r_i]\!]_1 = \mathsf{Dec}_{\mathsf{sk}}(m_{i,0}) = p \cdot q_i - \beta$. Therefore $r_i = p \cdot q_i$. In Step 5, each party will output $\sum_i [\![r_i]\!]_k = [\![\sum_i r_i]\!]_k$. Because $r_i = p \cdot q_i$ if and only if $y_i = x$ and 0 otherwise, this correctly results in shares of $[\![\sum_{x=y_i}(p \cdot q_i)]\!]$.

Security. For the security of our addition and deletion protocols, it suffices to show that the $\Pi_{\mathsf{CombinePath}}$ subroutine can be simulated against an adversarial $\mathsf{Responder}$. Let $\mathsf{View}_{\mathsf{R}}^{\mathsf{CP}}(x, p, \widetilde{\mathsf{path}}, \mathsf{sk}, \mathsf{pk})$ be the $\mathsf{Responder}$'s view, and we can construct a simulator $\mathsf{Sim}_{\mathsf{CP}}$ like so:

- In Step 1, $\mathsf{Sim}_{\mathsf{CP}}$ samples $\gamma_i \xleftarrow{\$} \mathbb{Z}_q$ for all $i \in [k]$ and sets $\mathsf{req}_i = \mathsf{Enc}_{\mathsf{pk}}(\gamma_i)$.
- In Step 4, $\mathsf{Sim}_{\mathsf{CP}}$ samples $\delta_i \leftarrow \mathbb{Z}_q$ for all $i \in [k]$ such that $\sum_i \delta_i = [\![z]\!]_1$ and sends $m_i = \mathsf{Enc}_{\mathsf{pk}}(\delta_i)$ to $\mathsf{Responder}$ on behalf of $\mathcal{F}_{\mathsf{lookup}}$.

Given that, we present the following lemma:

Lemma C.1. *In the $\mathcal{F}_{\mathsf{lookup}}$-hybrid model, there exists a PPT simulator $\mathsf{Sim}_{\mathsf{CP}}$ such that, for any inputs $x, p, \widetilde{\mathsf{path}}, \mathsf{sk}$, public key pk, and $\mathsf{Responder}$ output $[\![z]\!]_1$,*

$$\mathsf{View}_{\mathsf{R}}^{\mathsf{CP}}(x, p, \widetilde{\mathsf{path}}, \mathsf{sk}, \mathsf{pk}) \overset{c}{\approx} \mathsf{Sim}_{\mathsf{CP}}(1^\lambda, [\![z]\!]_1, \mathsf{pk}, k).$$

Proof. We prove the lemma with a hybrid argument:

Hyb_0: P_0's real view — i.e., $\mathsf{View}_{\mathsf{R}}^{\mathsf{CP}}(x, p, \widetilde{\mathsf{path}}, \mathsf{sk}, \mathsf{pk})$.

$\mathsf{Hyb}_{1,i}$: This is a series of hybrids where, in $\mathsf{Hyb}_{1,i}$, $m_{i,0}$ is replaced with a fresh encryption of $(p \cdot q_i - \beta_i)$. This is indistinguishable by the re-randomization property of the encryption scheme.

$\mathsf{Hyb}_{2,i}$: This is a series of hybrids where in $\mathsf{Hyb}_{2,i}$ $\mathsf{Sim}_{\mathsf{CP}}$ samples $\delta_i \xleftarrow{\$} \mathbb{Z}_q$ such that $\sum_{i=1}^k \delta_i = [\![z]\!]_1$ and sends $\mathsf{Enc}_{\mathsf{pk}}(\delta_i)$ to the $\mathsf{Responder}$ as the output of $\mathcal{F}_{\mathsf{lookup}}$. In the case where $m_i = m_{i,0}$, this is statistically indistinguishable because $p \cdot q_i - \beta_i$ has a uniform distribution over $\mathbb{Z}_q$ because β_i is sampled uniformly from it. In the case where $m_i = m_{i,1}$, this is indistinguishable because $-\beta_i$ has a uniform distribution over $\mathbb{Z}_q$ for the same reason.

$\mathsf{Hyb}_{3,i}$: This is a series of hybrids where, in $\mathsf{Hyb}_{3,i}$, SimCP samples $\gamma_i \xleftarrow{\$} \mathbb{Z}_q$ and sends $\mathsf{req}_i = \mathsf{Enc}_{\mathsf{pk}}(\gamma_i)$ in Step 2. This is statistically indistinguishable because $y_i - x + \alpha_i$ has a uniform distribution over $\mathbb{Z}_q$ because α_i is sampled uniformly from it.

The $\mathsf{Responder}$'s view in this hybrid is exactly $\mathsf{Sim}_{\mathsf{CP}}$'s output, concluding the proof.

□

D Proof of Theorem 4.3

Correctness. As before, we prove correctness by induction. On day 0, all sets are initialized as null sets so correctness trivially holds. Now, on any day d, our goal is to compute $I_d = \left(X \cup X_d^+ \setminus X_d^-\right) \cap \left(Y \cup Y_d^+ \setminus Y_d^-\right)$. First, similar to the addition only protocols in Section 3, from the correctness of the subroutines $\mathsf{UpdateTree}, \mathsf{GetPath}$ and subprotocol $\Pi_{\mathsf{CombinePath}}$, the additive homomorphic encryption scheme and the secret sharing scheme, one can observe that in the output generation, in Step 3c, the output computed is indeed using the reconstruction of the intersections listed in Steps Step 1b, Step 1c of the deletion phase and steps Step 2b, Step 2c of the addition phase. That is, in those 4 steps, after the reconstruction, we compute

$$I_1 = X_d^- \cap Y, \qquad I_2 = \left(X \setminus X_d^-\right) \cap Y_d^-,$$
$$I_3 = X_d^+ \cap \left(Y \setminus Y_d^-\right), \qquad I_4 = \left(X \cup X_d^+ \setminus X_d^-\right) \cap Y_d^+.$$

Elements in I_1 and I_2 are removed from I_d (since the reconstructed share is negative) and elements in I_3 and I_4 are added to I_d (since the reconstructed share is positive). That is, our protocol computes $I_d = ((I_{d-1} \cup I_3 \cup I_4) \setminus I_1) \setminus I_2$, where $I_{d-1} = X \cap Y$.

Recall that we assume no element can be added and deleted on the same day. That is, $X_d^+ \cap X_d^- = Y_d^+ \cap Y_d^- = \emptyset$. Now, let's expand I_d as follows:

$$\begin{aligned}
I_d &= \left(X \cup X_d^+ \setminus X_d^-\right) \cap \left(Y \cup Y_d^+ \setminus Y_d^-\right) \\
&= \left(\left(X \cup X_d^+ \setminus X_d^-\right) \cap \left(Y \setminus Y_d^-\right)\right) \cup \left(\left(X \cup X_d^+ \setminus X_d^-\right) \cap \left(Y_d^+\right)\right) \\
&= \left(\left(X \cup X_d^+ \setminus X_d^-\right) \cap \left(Y \setminus Y_d^-\right)\right) \cup I_4 \qquad \text{(by definition)} \\
&= \left(\left(X \setminus X_d^-\right) \cap \left(Y \setminus Y_d^-\right)\right) \cup \left(\left(X_d^+ \setminus X_d^-\right) \cap \left(Y \setminus Y_d^-\right)\right) \cup I_4 \\
&= \left(\left(X \setminus X_d^-\right) \cap \left(Y \setminus Y_d^-\right)\right) \cup I_3 \cup I_4 \qquad \text{(by definition since } X_d^+ \cap X_d^- = \emptyset)
\end{aligned}$$

Now, let's rewrite things a bit more. Observe that:

$$\begin{aligned}
X \cap Y &= \left(\left(X \setminus X_d^-\right) \cap \left(Y \setminus Y_d^-\right)\right) \cup \left(X_d^- \cap Y\right) \cup \left(\left(X \setminus X_d^-\right) \cap Y_d^-\right) \\
I_{d-1} &= \left(\left(X \setminus X_d^-\right) \cap \left(Y \setminus Y_d^-\right)\right) \cup I_1 \cup I_2
\end{aligned}$$

In other words, $\left(X \setminus X_d^-\right) \cap \left(Y \setminus Y_d^-\right) = (I_{d-1} \setminus I_1) \setminus I_2$. Putting this back into the first equation above, we get:

$$\begin{aligned}
I_d &= ((I_{d-1} \setminus I_1) \setminus I_2) \cup I_3 \cup I_4 \\
&= ((I_{d-1} \cup I_3 \cup I_4) \setminus I_1) \setminus I_2
\end{aligned}$$

Since $I_3 \cap I_1 = I_3 \cap I_2 = I_4 \cap I_1 = I_4 \cap I_2 = \emptyset$ (by definition and by the assumption that $X_d^+ \cap X_d^- = Y_d^+ \cap Y_d^- = \emptyset$). This concludes the proof of correctness.

Lemma D.1 (Corollary of Imported Lemma B.1)**.** *Let $\{x_i\}_{i=1}^N$ be a sequence of elements being added or removed to a binary tree via* $\mathsf{UpdateTree}$ *where any single element is added and removed at most t times. By increasing the node size σ and stash size ρ by a factor of* $\min(2t, \log N)$*, the probability of abort is negligible.*

Proof. By Imported Lemma B.1, abort is negligible after adding N elements with node size σ and stash size ρ. Let us first consider the case where we allow arbitrary additions or deletions. Note that because $\mathsf{UpdateTree}$ removes duplicates before placing elements into the tree, only a single additions or deletions for x_i will appear in any node. Additionally, $\mathsf{UpdateTree}$ keeps the invariant that additions and deletions for any element x_i in the tree will appear either in the root to leaf path $\mathcal{P}(F_{\mathsf{k}}(x_i))$ or in $\mathcal{S}$. Therefore, for any x_i, the maximum number of additions and deletions that can appear in the tree or stash is $\log N + 1$ — one for each node in $\mathcal{P}(F_{\mathsf{k}}(x_i))$ and one for $\mathcal{S}$. In the worse case, every element appears $\log N$ times in the tree and once in the stash, so they can be added to a tree with node size $\log N \cdot \sigma$ and stash size $\log N \cdot \rho$ without abort. In the case where elements can be added and removed at most t times and $2t < \log N$, in the worst case any element x_i will be duplicated in the tree $2t$ times. Therefore, a tree with node size $2t \cdot \sigma$ and stash size $2t \cdot \rho$ will suffice. □

Security against Corrupted P_0. Sim_0 can be constructed that simulates P_0's view as follows. Let $\mathcal{F}_0(X_{[D]}, Y_{[D]}) = \{Z_1, Z_2, \ldots, Z_D\}$ where Z_d is the set of elements added to the intersection on day d. On input $(1^\lambda, X_{[D]}, \mathcal{F}_0(X_{[D]}, Y_{[D]}))$, 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 the deletion phase Step 1c, Sim_0 runs $\mathsf{Sim}_{\mathsf{CP}}(1^\lambda, [\![z^-_{y,j}]\!]_0, \mathsf{pk}_0, k_d)$ to simulate P_0's view of $\Pi_{\mathsf{CombinePath}}$ for all $j \in [m^-]$ where $k_d = \sigma \cdot L_0 + \rho$.
- In the deletion phase Step 1d, Sim_0 samples a random set $Y'^-_d \xleftarrow{\$} \mathbb{Z}_q^{|Y^-_d|}$ and sends $(\{(\widetilde{\mathsf{updates}}_j, \ell_j)\}_{j=1}^{m^-}, \widetilde{\mathcal{S}}'_1) \leftarrow \mathsf{UpdateTree}(Y'^-_d, \{-y'_j : y'_j \in Y'^-_d\}_{j=1}^{m^-} \mathcal{D}_1, \mathcal{S}_1, \mathsf{pk}_1, F_{\mathsf{k}}(\cdot))$.
- In the addition phase Step 2c, Sim_0 runs $\mathsf{Sim}_{\mathsf{CP}}(1^\lambda, [\![z^+_{y,j}]\!]_0, \mathsf{pk}_0, k_d)$ to simulate P_0's view of $\Pi_{\mathsf{CombinePath}}$ for all $j \in [m^+]$ where $k_d = \sigma \cdot L_0 + \rho$.
- In the addition phase Step 2d, Sim_0 samples a random set $Y'^+_d \xleftarrow{\$} \mathbb{Z}_q^{|Y^+_d|}$ and sends $(\{(\widetilde{\mathsf{updates}}_j, \ell_j)\}_{j=1}^{m^+}, \widetilde{\mathcal{S}}'_1) \leftarrow \mathsf{UpdateTree}(Y'^+_d, \{y'_j : y'_j \in Y'^+_d\}_{j=1}^{m^+} \mathcal{D}_1, \mathcal{S}_1, \mathsf{pk}_1, F_{\mathsf{k}}(\cdot))$.
- In the output generation phase Step 3, Sim_0 does the following.
  - In Step 3b, for $1 \leq i \leq |Z_d|$, Sim_0 sets $[\![z_i]\!]_0 = 0$ and $[\![z_i]\!]_1 = z_i$ where $z_i \in Z_d$. For $|Z_d| \leq i \leq \Gamma$, Sim_0 samples shares of zero uniformly ($[\![z_i]\!]_0 \xleftarrow{\$} \mathbb{Z}_q$ and $[\![z_i]\!]_1 = -[\![z_i]\!]_0$).
  - In Step 3c, Sim_0 encrypts the all $[\![z_i]\!]_0$ set in Step 3b and uses that in their response.

Finally, Sim_0 outputs P_0's view. Using the below hybrid argument, we show that the real and ideal worlds are indistinguishable.

Hyb_0: This is the real world.

Hyb_1: This is same as Hyb_0 except that the message in Step 3b is computed using the shares set by Sim_0. Since $\{\alpha_i\}_{i \in [N]}$ are randomly sampled and π is a random permutation, the set of shares that P_0 learns are identically distributed in both hybrids. Hence, they are statistically indistinguishable.

Hyb_2: Same as Hyb_1 except that P_1 never aborts in Step 2d of $\mathsf{UpdateTree}$. This hybrid computationally indistinguishable from Hyb_1 because of the pseudorandomness of $F_{\mathsf{k}}(\cdot)$. If one can distinguish between Hyb_1 and Hyb_2, then it means the abort probability in Hyb_2 is non-negligible. By Lemma D.1, if the function $F(\cdot)$ used in $\mathsf{UpdateTree}$ is a random function, then the probability of abort is negligible. Hence we can use the abort probability to distinguish between a pseudorandom function $F_{\mathsf{k}}(\cdot)$ and a random function $F(\cdot)$.

Hyb_3: Same as Hyb_2 except in the addition phase Step 2d, Y_d^+ is replaced with $Y_d'^+$ in $\mathsf{UpdateTree}$. By the construction of $\mathsf{UpdateTree}$ and CPA security of the encryption scheme, this hybrid is computationally indistinguishable.

$\mathsf{Hyb}_{4,i}$: This is a series of hybrids where, in $\mathsf{Hyb}_{4,i}$, P_0's view of the ith $\Pi_{\mathsf{CombinePath}}$ in Step 2c is simulated with $\mathsf{Sim}_{\mathsf{CP}}(1^\lambda, [\![z_{y,i}^+]\!]_0, \mathsf{pk}_0, k_d)$. This is computationally indistinguishable by Lemma C.1. Let Hyb_4 be the last hybrid in this series of hybrids.

Hyb_5: Same as Hyb_4 except that P_1 never aborts in Step 2d of $\mathsf{UpdateTree}$. This hybrid computationally indistinguishable from Hyb_4 because of the pseudorandomness of $F_{\mathsf{k}}(\cdot)$ and Lemma D.1.

Hyb_6: Same as Hyb_5 except in the addition phase Step 1d, Y_d^- is replaced with $Y_d'^-$ in $\mathsf{UpdateTree}$. By the construction of $\mathsf{UpdateTree}$ and CPA security of the encryption scheme, this hybrid is computationally indistinguishable.

$\mathsf{Hyb}_{6,i}$: This is a series of hybrids where, in $\mathsf{Hyb}_{6,j}$, P_0's view of the jth $\Pi_{\mathsf{CombinePath}}$ in Step 1c is simulated with $\mathsf{Sim}_{\mathsf{CP}}(1^\lambda, [\![z_{y,j}^-]\!]_0, \mathsf{pk}_0, k_d)$. This is computationally indistinguishable by Lemma C.1.

P_0's view in the last hybrid of this series is exactly Sim_0's output, concluding the proof.

Security Against Corrupted P_1. Sim_1 can be constructed that simulates P_1's view as follows. On input $(1^\lambda, Y_{[D]})$, 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 the deletion phase Step 1a, Sim_1 samples a random set $X_d'^- \xleftarrow{\$} \mathbb{Z}_q^{|X_d^-|}$ and sends $(\{(\widetilde{\mathsf{updates}}_i, \ell_i)\}_{i=1}^{n^-}, \widetilde{\mathcal{S}}_0') \leftarrow \mathsf{UpdateTree}(X_d'^-, \{-x_i' : x_i' \in X_d'^-\}_{i=1}^{n^-} \mathcal{D}_0, \mathcal{S}_0, \mathsf{pk}_0, F_{\mathsf{k}}(\cdot))$.
- In the deletion phase Step 1b, Sim_1 runs $\mathsf{Sim}_{\mathsf{CP}}(1^\lambda, [\![z_{x,i}^-]\!]_1, \mathsf{pk}_1, k_d)$ to simulate P_1's view of $\Pi_{\mathsf{CombinePath}}$ for all $i \in [n^-]$ where $k_d = \sigma \cdot L_1 + \rho$.
- In the addition phase Step 2a, Sim_1 samples a random set $X_d'^+ \xleftarrow{\$} \mathbb{Z}_q^{|X_d^+|}$ and sends $(\{(\widetilde{\mathsf{updates}}_i, \ell_i)\}_{i=1}^{n^+}, \widetilde{\mathcal{S}}_0') \leftarrow \mathsf{UpdateTree}(X_d'^+, \{x_i' : x_i' \in X_d'^+\}_{i=1}^{n^+} \mathcal{D}_0, \mathcal{S}_0, \mathsf{pk}_0, F_{\mathsf{k}}(\cdot))$.
- In the addition phase Step 2b, Sim_1 runs $\mathsf{Sim}_{\mathsf{CP}}(1^\lambda, [\![z_{x,i}^+]\!]_1, \mathsf{pk}_1, k_d)$ to simulate P_1's view of $\Pi_{\mathsf{CombinePath}}$ for all $i \in [n^+]$ where $k_d = \sigma \cdot L_1 + \rho$.
- In the output generation phase Step 3a, Sim_1 samples random shares $[\![z_i]\!]_0$ for $i \in [\Gamma]$ and sends $\{\mathsf{Enc}_{\mathsf{pk}_0}([\![z_i]\!]_0)\}_{i=1}^{\Gamma}$ to P_1.

Finally, Sim_1 outputs P_1's view. Using the below hybrid argument, we show that the real and ideal worlds are indistinguishable.

Hyb_0: This is the real world.

Hyb_1: Same as Hyb_0 except that in the output generation phase Step 3a, the ciphertexts $\mathsf{Enc}_{\mathsf{pk}_0}([\![z_i]\!]_0)$ is a tuple of fresh encryptions of random values. This is actually a series of sub-hybrids where each ciphertext is replaced in each sub-hybrid. These are computationally indistinguishable by the CPA security of the encryption scheme.

$\mathsf{Hyb}_{2,i}$: This is a series of hybrids where, in $\mathsf{Hyb}_{2,i}$, P_1's view of the ith $\Pi_{\mathsf{CombinePath}}$ in Step 2b is simulated with $\mathsf{Sim}_{\mathsf{CP}}(1^\lambda, [\![z_{x,i}^+]\!]_1, \mathsf{pk}_1, k_d)$. This is computationally indistinguishable by Lemma C.1. Let Hyb_2 be the last hybrid in this series of hybrids.

Hyb_3: Same as Hyb_2 except that P_1 never aborts in Step 2d of $\mathsf{UpdateTree}$. This hybrid computationally indistinguishable from Hyb_2 because of the pseudorandomness of $F_{\mathsf{k}}(\cdot)$ and Lemma D.1.

Hyb_4: Same as Hyb_3 except in the addition phase Step 2a, X_d^+ is replaced with $X_d'^+$ in $\mathsf{UpdateTree}$. By the construction of $\mathsf{UpdateTree}$ and CPA security of the encryption scheme, this hybrid is computationally indistinguishable.

$\mathsf{Hyb}_{4,i}$: This is a series of hybrids where, in $\mathsf{Hyb}_{4,i}$, P_1's view of the ith $\Pi_{\mathsf{CombinePath}}$ in Step 1b is simulated with $\mathsf{Sim}_{\mathsf{CP}}(1^\lambda, [\![z_{x,i}^-]\!]_1, \mathsf{pk}_1, k_d)$. This is computationally indistinguishable by Lemma C.1. Let Hyb_4 be the last hybrid in this series of hybrids.

Hyb_5: Same as Hyb_4 except that P_1 never aborts in Step 2d of $\mathsf{UpdateTree}$. This hybrid computationally indistinguishable from Hyb_4 because of the pseudorandomness of $F_{\mathsf{k}}(\cdot)$ and Lemma D.1.

Hyb_6: Same as Hyb_5 except in the addition phase Step 1a, X_d^- is replaced with $X_d'^-$ in $\mathsf{UpdateTree}$. By the construction of $\mathsf{UpdateTree}$ and CPA security of the encryption scheme, this hybrid is computationally indistinguishable.

P_0's view in this hybrid is exactly Sim_0's output, concluding the proof.

E Proof of Theorem 4.2

Correctness. As before, we prove correctness by induction. On day 0, all sets are initialized as null sets so correctness trivially holds. Now, on any day d, let's define a function $f(x_i, y_j) = p_i \cdot q_j$ where for any $x_i \in (X \cup X_d^+ \cup X_d^-)$, $y_j \in (Y \cup Y_d^+ \cup Y_d^-)$ (p_i, q_j) are defined as in the protocol. Then, as in the correctness of the previous protocol, by definition, observe that, for both functionalities:

$$\begin{aligned}\mathsf{Out}_d &= \sum_{\substack{x_i \in (X \cup X_d^+ \cup X_d^-),\\ y_j \in (Y \cup Y_d^+ \cup Y_d^-)}} f(x_i, y_j)\\ &= \sum_{\substack{x_i \in X,\\ y_j \in Y}} f(x_i, y_j) + \sum_{\substack{x_i \in (X_d^+ \cup X_d^-),\\ y_j \in Y}} f(x_i, y_j) + \sum_{\substack{x_i \in (X \cup X_d^+ \cup X_d^-),\\ y_j \in (Y_d^+ \cup Y_d^-)}} f(x_i, y_j).\end{aligned}$$

We can observe the following in the protocol from the correctness of the underlying primitives:

- In Step 3, the outputs are secret shares of $\sum_{\substack{x_i \in (X_d^+ \cup X_d^-),\\ y_j \in Y}} f(x_i, y_j)$.
- In Step 4, the outputs are secret shares of $\sum_{\substack{x_i \in (X \cup X_d^+ \cup X_d^-),\\ y_j \in (Y_d^+ \cup Y_d^-)}} f(x_i, y_j)$.

and by induction, $\mathsf{Out}_{d-1} = \sum_{\substack{x_i \in X,\\ y_j \in Y}} f(x_i, y_j)$. Then, by the correctness of the reconstruction of the secret sharing scheme, Out_d is correctly computed and this completes the proof.

Security Against Corrupted P_0. Sim_0 can be constructed that simulates P_0's view as follows. On input $(1^\lambda, X_{[D]}, \mathcal{F}_0(X_{[D]}, Y_{[D]}))$, 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 4, Sim_0 runs $\mathsf{Sim}_{\mathsf{CP}}(1^\lambda, [\![z_{y,j}]\!]_0, \mathsf{pk}_0, k_d)$ to simulate P_0's view of $\Pi_{\mathsf{CombinePath}}$ for all $j \in [m]$ where $k_d = \sigma \cdot L_0 + \rho$.

- In Step 5, Sim_0 samples a random set $(Y_d'^- \cup Y_d'^+) \xleftarrow{\$} \mathbb{Z}_q^{|Y_d^-|+|Y_d^+|}$ and sends $(\{(\widetilde{\mathsf{updates}}_j, \ell_j)\}_{j=1}^m, \widetilde{\mathcal{S}}_1') \leftarrow \mathsf{UpdateTree}(Y_d'^- \cup Y_d'^+, \{q_j'\}_{j=1}^m \mathcal{D}_1, \mathcal{S}_1, \mathsf{pk}_1, F_\mathsf{k}(\cdot))$ where q_j' is defined as in the real world.
- In the output generation phase Step 7, Sim_0 does the following.
  - First, let $\mathcal{F}_0(X_{[D]}, Y_{[D]}) = \{\mathsf{Out}_1, \mathsf{Out}_2, \ldots, \mathsf{Out}_D\}$ where Out_d is different for $\mathcal{F}_{\mathsf{UPSI\text{-}Del_{ca}}}$ and $\mathcal{F}_{\mathsf{UPSI\text{-}Del_{sum}}}$ respectively.
  - Since Sim_0 knows P_0's input (and randomness), it can compute the value $[\![z_d]\!]_0$ that P_0 would have computed in Step 6.
  - Sim_0 computes and sends $[\![z_d]\!]_1 = \mathsf{Out}_d - \mathsf{Out}_{d-1} - [\![z_d]\!]_0$.

Finally, Sim_0 outputs P_0's view. Using the below hybrid argument, we show that the real and ideal worlds are indistinguishable.

Hyb_0: This is the real world.

Hyb_1: This is same as Hyb_0 except that the the output generation phase Step 7 happens as in the ideal world. That is, $[\![z_d]\!]_1$ is set as $(\mathsf{Out}_d - \mathsf{Out}_{d-1} - [\![z_d]\!]_0)$ where $[\![z_d]\!]_0$ is P_0's share computed in Step 6.

From the correctness of $\Pi_{\mathsf{CombinePath}}$ and security of the secret sharing scheme, the share $[\![z_d]\!]_1$ that P_0 learns is identically distributed in both hybrids and so are statistically indistinguishable.

Hyb_2: Same as Hyb_1 except that P_1 never aborts in Step 2d of $\mathsf{UpdateTree}$. This hybrid computationally indistinguishable from Hyb_4 because of the pseudorandomness of $F_\mathsf{k}(\cdot)$ and Lemma C.1.

Hyb_3: Same as Hyb_2 except in Step 5, $(Y_d^+ \cup Y_d^-)$ is replaced with $(Y_d'^- \cup Y_d'^+)$ in $\mathsf{UpdateTree}$. By the construction of $\mathsf{UpdateTree}$ and CPA security of the encryption scheme, this hybrid is computationally indistinguishable.

$\mathsf{Hyb}_{4,i}$: This is a series of hybrids where, in $\mathsf{Hyb}_{4,i}$, P_0's view of the ith $\Pi_{\mathsf{CombinePath}}$ in Step 4 is simulated with $\mathsf{Sim}_{\mathsf{CP}}(1^\lambda, [\![z_{y,i}]\!]_0, \mathsf{pk}_0, k_d)$. This is computationally indistinguishable by Lemma C.1.

P_0's view in the last hybrid of this series is exactly Sim_0's output, concluding the proof.

Security Against Corrupted P_1. Sim_1 can be constructed that simulates P_1's view as follows. On input $(1^\lambda, Y_{[D]})$, 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, Sim_1 samples a random set $(X_d'^- \cup X_d'^+) \xleftarrow{\$} \mathbb{Z}_q^{|X_d^-|+|X_d^+|}$ and sends $(\{(\widetilde{\mathsf{updates}}_i, \ell_i)\}_{i=1}^n, \widetilde{\mathcal{S}}_0') \leftarrow \mathsf{UpdateTree}(X_d'^- \cup X_d'^+, \{p_i'\}_{i=1}^n \mathcal{D}_0, \mathcal{S}_0, \mathsf{pk}_0, F_\mathsf{k}(\cdot))$ where p_i' is defined as in the real world.
- In Step 3, for each run of $\Pi_{\mathsf{CombinePath}}$, while setting its input, Sim_1 replaces (x_i, p_i) by sampling a random tuple (x_i', p_i'), random values $\alpha_j, \beta_j \xleftarrow{\$} \mathbb{Z}_q$ for $1 \le j \le |l_d|$ and sets $\widetilde{\mathsf{path}}_i = \{(\mathsf{Enc}_{\mathsf{pk}_1}(\alpha_j), \mathsf{Enc}_{\mathsf{pk}_1}(\beta_j))\}_{j=1}^{l_d}$ where $l_d = \sigma \cdot L_1 + \rho$.

Finally, Sim_1 outputs P_1's view. Using the below hybrid argument, we show that the real and ideal worlds are indistinguishable.

Hyb_0: This is the real world.

$\mathsf{Hyb}_{1,i}$: This is a series of hybrids where, in $\mathsf{Hyb}_{1,i}$, P_1's view of the ith $\Pi_{\mathsf{CombinePath}}$ in Step 3 is simulated with $\mathsf{Sim}_{\mathsf{CP}}(1^\lambda, [\![z_{x,i}]\!]_1, \mathsf{pk}_1, k_d)$. This is computationally indistinguishable by Lemma C.1. Let Hyb_1 be the last hybrid in this series of hybrids.

Hyb_2: Same as Hyb_1 except that P_1 never aborts in Step 2d of $\mathsf{UpdateTree}$. This hybrid computationally indistinguishable from Hyb_1 because of the pseudorandomness of $F_{\mathsf{k}}(\cdot)$ and Lemma C.1.

Hyb_3: Same as Hyb_2 except in Step 2, $(X_d^+ \cup X_d^-)$ is replaced with $(X_d'^- \cup X_d'^+)$ in $\mathsf{UpdateTree}$. By the construction of $\mathsf{UpdateTree}$ and CPA security of the encryption scheme, this hybrid is computationally indistinguishable.

P_0's view in this hybrid is exactly Sim_0's output, concluding the proof.