Updatable Private Set Intersection from Structured Encryption

Archita Agarwal
MongoDB
archita.agarwal@mongodb.com

David Cash
University of Chicago
davidcash@uchicago.edu

Marilyn George
MongoDB
marilyn.george@mongodb.com

Seny Kamara
MongoDB & Brown University
seny.kamara@mongodb.com

Tarik Moataz
MongoDB
tarik.moataz@mongodb.com

Jaspal Singh*
Purdue University
sing1361@purdue.edu

July 22, 2024

Abstract

Many efficient custom protocols have been developed for two-party private set intersection (PSI), that allow the parties to learn the intersection of their private sets. However, these approaches do not yield efficient solutions in the dynamic setting when the parties' sets evolve and the intersection has to be computed repeatedly. In this work we propose a new framework for this problem of *updatable PSI* — with elements being inserted and deleted — in the semi-honest model based on structured encryption. The framework reduces the problem of updatable PSI to a new variant of structured encryption (StE) for an updatable set datatype, which may be of independent interest. Our final construction is a constant round protocol with worst-case communication and computation complexity that grows linearly in the size of the updates and only poly-logarithmically with the size of the accumulated sets. Our protocol is the first to support arbitrary inserts and deletes for updatable PSI.

1 Introduction

Private set intersection (PSI) protocols allow two parties with input sets A and B respectively, to learn the intersection $A \cap B$, while hiding each input set from the other party. Efficient custom protocols have been developed for two party PSI based on public-key primitives [DT10, JL10, RT21], oblivious transfer extension [PSSZ15, KKRT16, RR17, PRTY19, PRTY20, CM20, GPR+21, RS21] and vector oblivious linear evaluation [RS21, CILO22], where both the communication and the computation complexity of the protocol scale linearly or almost linearly with the size of the input sets. Protocols for PSI and related private set operations have been used in a number of privacy-preserving applications, including online advertisement [IKN+20], contact discovery [DRRT18, KRS+19, HSW23], and public-key authentication for SSH [RLJR22].

Updatable PSI. For a number of applications of PSI including online advertisement [IKN+20] and password breach monitoring [MIC], the set intersection is computed multiple times as the sets grow or shrink over time. This notion of *Updatable PSI* was first formalized by Badrinarayanan et

*Work done while at MongoDB

al. [BMX22]. The authors proposed two protocols based on the Decisional Diffie-Hellman (DDH) assumption, where the complexity of successive PSI computations is linearly dependent on only the size of the updates and not the size of the entire input sets. Their first protocol only supports inserts, and the second protocol supports inserts along with a weak notion of deletes — inserted elements can only be deleted after a certain number of epochs.

Arbitrary deletions. A protocol for updatable PSI that supports arbitrary deletions is not known to date; but it would be a valuable tool for privacy-preserving applications. Consider for example, the application of measuring online ad statistics [IKN+20]. In this setting, we have two parties: a merchant running an online ad campaign, and an online ad agency offering a platform where users can interact with the merchant's ads. The merchant is interested in measuring the effectiveness of their ad campaign over a period of time. This would involve computing some statistics (including functions of the set intersection) over the user data of both the merchant and the ad agency. These aggregate statistics would have to be computed repeatedly over a period of time. In order for both the merchant and the ad agency to stay compliant with privacy laws (like GDPR), they must be able to update their data, including inserting or deleting user records. Hence, a key building block for such a privacy-preserving application would be an efficient protocol for computing private set intersection and related functionalities (like union or cardinality of the intersection) with the ability to update sets arbitrarily over time.

This leads to the following natural question, which we affirmatively answer in this work:

> *Can we design updatable PSI protocols that support arbitrary insertions and deletions in constant rounds and with communication and computation complexity that is sublinear in the size of the accumulated sets?*

1.1 Our Contributions

In this paper we construct such an updatable PSI protocol, where either party can insert or delete elements. Our protocol scales with the sizes of the parties' updates, and only poly-logarithmically with the size of their accumulated sets. Our construction stems from a general framework that builds updatable PSI (with arbitrary deletions) generically from a flavor of *dynamic structured encryption (StE)* [CGKO06, CK10] for the set data structure. Dynamic StE is a cryptographic primitive that allows a client to create, query, and update an encrypted data structure stored on an untrusted server.

A framework for UPSI. Our framework requires the underlying StE scheme to support updates from the client as well as membership queries from the (untrusted) server.[1] At a high level, our framework generalizes the Elgamal encryption based updatable PSI construction of Badrinarayanan et al. [BMX22] that supports only insertions and weak deletions.

Each party (as an untrusted server) holds an encrypted data structure that represents the other party's (client's) current set. First, both parties use the underlying StE scheme to update their own sets. This is followed by invocations of the "server-side" query of the StE scheme as well as a generic private set union (PSU) protocol to reveal the new intersection. Our framework is general, in that it can use any dynamic StE for sets that supports server-side querying to generate an updatable PSI protocol. However, the leakage of the resulting protocol will vary depending on the leakage of the underlying StE scheme and PSU protocol. In order to enable its general use, we formalize the exact leakage of our updatable PSI protocol in terms of the leakage of StE and PSU.

[1] In contrast to traditional dynamic StE, which only requires a scheme to support updates and queries from a trusted client.

Our approach generalizes prior work, but in implementing it we must confront two difficulties, one algorithmic and the other definitional: First, the needed StE does not exist in the literature, and there are technical challenges in realizing it while maintaining *minimal leakage* in the updatable PSI framework, where only the sizes of the update sets are revealed in each epoch to both the parties. Second, this notion of minimal leakage is difficult to capture with standard 2pc definitions [Lin17].

ESX: A DYNAMIC STE FOR SETS. After our framework, our main technical contribution is the design of a dynamic StE scheme ESX that can be used with the framework and may be of independent interest. We start by designing a traditional StE scheme that only supports queries and updates from the client. This scheme leaks the query equality to the server for membership queries, but has minimal leakage for updates. Its protocols are constant round and it scales poly-logarithmically with the size of the set. As we discuss in our technical overview, this requires new insights for ORAM-like tree data structures that can change size over time.

ESX WITH SERVER-SIDE QUERYING. In order to use ESX in our framework for updatable PSI, we require the novel functionality of *server-side querying*. In particular, the party holding the encrypted set structure (representing the other party's set) has to be able to execute membership queries over the encrypted set. We then modify ESX to support server-side querying with similar asymptotic complexity and *minimal leakage* for both updates and server-side queries. We note that server-side querying has not been considered in the prior StE literature, and might be of independent interest.

OUR UPDATABLE PSI: INSTANTIATING THE FRAMEWORK. Our construction of ESX with server-side querying can be instantiated with an OPRF protocol based on alternating-moduli PRFs due to Alamati et. al [APRR24] and a generic 2pc protocol like garbled circuits due to Rosuelek and Roy [RR21]. This construction, along with the PSU protocol of Zhang et al. [ZCL$^+$23] or Bienstock et al. [BPSY23] can be used to instantiate our framework; resulting in an updatable PSI protocol that supports arbitrary inserts and deletes with *minimal leakage* - i.e., the protocol leaks only the size of the update sets in each epoch. Further, for each epoch, our protocol takes constant rounds, and has worst-case communication and computation complexity that scales linearly with the size of the update sets up to poly-logarithmic factors.

2PC WITH LEAKAGE. In order to accurately describe the security of our updatable PSI framework and protocol, we also introduce general definitions of 2pc with leakage. A typical definition of 2pc security requires, informally, that "nothing is revealed to either party, beyond what they can compute from their own input and output". The precise meaning of this security guarantee can be hard to interpret. Specifically, when a 2pc protocol (and its target functionality) assume that inputs are of a certain size, or fit a given format, then arguably this size/format information is being revealed. For example, Badrinarayanan et al. assume that each party wishes to add a fixed number of elements in each epoch, or is willing to pad their additions up to that fixed number. In practice, larger additions would require multiple runs of the protocol, effectively leaking information on the size of the updates.

In this paper we take a generalized view of 2pc, where we allow for functionalities that accept inputs of any size or type. As a result, we must also allow explicit *leakage* that is given to the simulator, in order to express the information revealed about the size and type of the inputs. In particular, in the case of an updatable PSI protocol, our functionality allows sets of any size to be input, and the corresponding leakage explicitly states the information that will be revealed to the parties.[2] This approach has some downsides, like added complexity (especially to composition),

[2]This definitional approach also results in our *minimal leakage* being the size of the updates during the protocol. The alternative would be to assume that the functionality only accepts updates of a fixed size which is known to

but we argue that this approach can be used for giving security theorems that more closely match applications.

1.2 Technical Overview

We now highlight the key technical ideas and challenges in our work. Section 2 presents our security definitions and other preliminaries. Section 3 describes our general framework for updatable PSI based on dynamic StE, Section 4 describes our construction of the dynamic StE scheme ESX, and Section 5 describes our final instantiation of the updatable PSI protocol.

Our framework: UPSI from dynamic StE. Our framwork requires a dynamic StE scheme with server-side querying which is used to create, update, and server-side query the encrypted sets. Let our parties be P_X and P_Y with input sets X and Y. Let X_+ and X_- be the elements that P_X wants to add and delete from set X, and similarly Y_+ and Y_- for P_Y. Given the existing intersection $I_0 = X \cap Y$, for one epoch of updates, notice that the updated intersection

$$I_1 = (I_0 \setminus W) \cup U,$$

where $W = (X_- \cap I_0) \cup (Y_- \cap I_0)$, $U = (Y_+ \cap X_1) \cup (X_+ \cap Y_1)$, and X_1 and Y_1 are the updated sets X and Y. Our framework computes the sets U and W, and the parties can then compute the updated intersection locally. In our framework, each party holds an encrypted data structure that represents the other party's current set, and proceeds as follows:

- Set U: elements to be added to the current intersection. P_X first updates the encrypted set X to X_1 (held by P_Y). After the updates, P_Y runs the server-side membership query protocol on the encrypted set X_1 to compute $(Y_+ \cap X_1)$. By the symmetric process, P_X computes $(X_+ \cap Y_1)$. The parties then use a PSU protocol to compute the set U.

- Set W: elements to be removed from the current intersection. P_X computes $(X_- \cap I_0)$ locally, and similarly P_Y computes $(Y_- \cap I_0)$. They then use a PSU protocol to compute the set W.

Our framework is depicted in Figure 6, and the security of the resulting updatable PSI protocol is proved as Theorem 3.1. From our security theorem, in order to obtain an updatable PSI protocol with minimal leakage, we have to design a dynamic StE scheme for minimal leakage. Since our framework uses only client updates and server-side queries, our main goal will be to unlink updates and queries in the StE scheme.

Our dynamic StE: Client-side query. As a first step, we give a traditional StE version of the construction, where the client inputs both the updates and queries. Our construction utilizes an "ORAM-like" tree with log-size buckets but without a recursive position map. Querying for elements of the sets simply involves evaluating a PRF to determine a path, requesting that path from the server and checking it for the relevant element. Hence, querying the same element fetches the same path from the tree — leaking *query equality* to the server, unlike a typical tree ORAM.

Updates are more technically involved. The main challenge is that the underlying set is growing and shrinking, while we would like updates (adds or deletes) and queries to not reveal information about each other. To unlink updates and queries, we use the ORAM approach of adding elements to the root of the tree and letting oblivious evictions eventually move them down the tree. Further, we perform deletions lazily, meaning that to delete x, we add a flag indicating that x should be

both parties.

deleted.[3] While deletes temporarily consume more space, they will eventually be cleaned up during evictions.

The main technical novelty in our construction is the management of the size of the tree with minimal leakage. As data is added and deleted, we gradually add and delete leaves of the tree to change its overall capacity. This is a delicate process because of how it interacts with our lazy deletions: Since those deletions consume more space temporarily, it is not the size of the set, but the number of "slots" used in the tree that should determine the capacity. This number will vary depending on how many deletes are cleaned up during evictions.

However, the decision to grow or shrink the tree is visible to the server, and a naive approach will result in unintended leakage. For example, if evictions opportunistically lower the size of the tree too early and cause us to start deleting leaves, then the server can infer that is it more likely we were adding and deleting the same element multiple times. We resolve this by growing and shrinking based on leaked information, namely only the total number of adds and deletes (but not *what* was added and deleted).

Our dynamic StE: Server-side query. To perform membership queries in this StE scheme, the server must be able to identify the path corresponding to an element, decrypt the path, and test for membership, all with minimal leakage. In order to fetch the correct path, the server and the client can run any oblivious PRF protocol. Decryption and membership testing can be done in secure 2pc to reveal only the final output to the server. We show that the resulting StE scheme has *no leakage beyond the size of the update and query sets.* Finally, we use this StE scheme with server-side querying to instantiate our updatable PSI protocol with minimal leakage.

1.3 Related Work

Conventional PSI. Over the last decade, the design of two-party and multi-party PSI protocols has been an active area of research, where the focus has been on developing concretely efficient solutions for different network settings and practical set sizes. There are several protocol paradigms for PSI, including circuit-based [HEK12, PSWW18], key agreement [DT10, JL10], oblivious transfer extension [PSSZ15, KKRT16, RR17, PRTY20, CM20, GPR$^+$21, RS21, BPSY23] and vector OLE [RS21, CILO22], to name a few. Most of these conventional protocols have computation and communication complexity that scale linearly with the size of the input sets. All these constructions leak the size of both input sets, along with the expected output (which is either one-sided or two-sided).

Sublinear Communication PSI. In the case where the input sets have asymmetric sizes, it is possible to construct two-party PSI solutions where the communication scales with the size of the smaller set. These solutions include those based on RSA accumulators [ADT11], pairing based accumulators [ALOS22], leveled fully homomorphic encryption [CLR17, CHLR18] and Computational Diffie-Hellman [ABD$^+$21]. All these protocols use expensive cryptographic operations (public-key operations), and they have linear computation overhead in the size of the larger set, making them not ideal for the updatable setting even when considering asymmetric set sizes.

For the asymmetric case, a number of works have also designed PSI solutions in the offline-online model, where in the offline phase the parties do some pre-processing given as input the larger set [RA18, KRS$^+$19]. In these constructions the online phase has computation and communication complexity that scales linearly with the size of the smaller set. However, these solutions have not

[3] In the actual construction, we just add x again to delete it. At query time, we check if x appears an even or odd number of times and determine if it is still in the set. Since both adds and deletes are now essentially the same, lazy deletion also helps us reduce the leakage of the resulting construction.

been explored in the updatable setting with one exception. Kiss et al. [KLS+17] extend their offline-online PSI framework to support insert and delete updates as well. However, their protocol has leakage beyond the size of the input and update sets. Particularly, when an element is output from their PSI protocol, both parties learn in which epoch the same element was previously inserted in the other party's set. In our updatable PSI framework we will avoid this 'historical' leakage using our novel StE construction, while paying a poly-logarithmic overhead in complexity.

Another new direction in PSI literature is to consider settings where one party's input set has a publicly known structure, allowing for more efficient PSI solutions where the communication scales with the description size of the structured set instead of its cardinality [GRS22, GRS23, GGM24]. This structure-aware PSI construction is based on oblivious transfer (OT) and some variant of function secret sharing (FSS). These protocols are especially useful to compute fuzzy PSI — where the two parties have elements in the intersection even if their points are ϵ close in some metric space. For this application the publicly known structure is a union of constant radius $\ell-$infinity balls. These solutions are only known for special types of structured sets, and they are not comparable to our updatable PSI solution for arbitrary sets.

Private Set Operations with updates. The reactive functionality of updatable PSI was first formulated by Badrinarayanan et al. [BMX22]. They developed two solutions based on the DDH assumption for updatable PSI, one that supports arbitrary inserts, and one for arbitrary inserts along with "weak deletion". Here weak deletion implies that elements inserted before the latest t epochs are deleted (where t is a parameter). Their constructions only leak the size of the update sets in each epoch, unlike the updatable PSI construction due to Kiss et al. [KLS+17]. Their solutions are also asymptotically optimal - with their communication and computation scaling linearly with the size of the update sets. Our new framework for updatable PSI improves on [BMX22] by allowing for *arbitrary deletes and inserts* in each epoch, at the cost of a poly-logarithmic overhead in computation and communication complexity. All our stated complexities are also worst-case, whereas [BMX22] costs are amortized over a larger number of epochs for weak deletions.

Dittmer et al. [DIL+22] study a weighted variant of asymmetric and updatable PSI in which the output is the sum of the weights of keywords in the intersection. Their approach avoids expensive public key cryptography, and instead uses symmetric key based FSS for point functions as the key building block. The communication complexity of each update and weighted-sum PSI computation scales linearly with the size of the updates, however the computation complexity of their protocol still scales with the size of the entire set. Their work is also limited to the three-party setting, where the client inputs the smaller set, and the larger input set is available with two non-colluding servers - making their model incompatible with ours.

Structured Encryption. Structured encryption (StE) was introduced by Chase and Kamara [CK10] as a generalization of index-based searchable symmetric encryption (SSE) [SWP00, CGKO06]. The most common and important type of StE schemes are multi-map encryption schemes which are a basic building block in the design of efficient SSE schemes [CGKO06, KPR12, CJJ+14], expressive SSE schemes [CJJ+13, FJK+15, KM17, KM18] and encrypted databases [KM18, CNR21]. StE and encrypted multi-maps have been studied along several dimensions including dynamism [KPR12, KP13, CJJ+14, HK14], I/O efficiency [CJJ+14, CT14, ANSS16, MM16, DP17, ASS21, DPP18], and for different security notions [Bos16, GMP16a, KKL+17, BMO17, EKPE18, SDY+18, AKM21].

The key building block in our updatable PSI construction is a dynamic StE scheme with server-side querying. To the best of our knowledge there is no prior work on structured encryption—except

for [KM22][4]—that focuses on server-side querying. Our StE construction is based on "ORAM-like" trees, which are resizable and have no position map. Our StE design choices ensure that the construction allows for updatable sets, and that it has constant round and worst-case polylogarithmic update/query complexity. We next describe some standard ORAM constructions, and discuss why they were not a good fit for our setting.

Oblivious RAM. ORAM allows a client to hide its data access patterns from an untrusted server that it uses for outsourcing data. This notion was first introduced by Goldreich and Ostrovsky [GO96], but it has since been heavily optimized for a number of applications [PR10, SDS+18, WCS15, Ds17]. Our newly proposed dynamic StE construction follows the tree-based ORAM paradigm [SDS+18, WCS15, DvDF+16] and specifically our eviction algorithm closely follows the eviction algorithm in Onion ORAM [DvDF+16]. All standard tree-based ORAM constructions support fixed array size and have logarithmic round complexity, and hence they cannot be directly used to design our StE construction. TWORAM [GMP16b] does avoid the logarithmic round complexity using server-side computation and by employing garbled circuit based gadgets to offload most of the computation to the server. However by default this construction does not support dynamic arrays — which is critical for our updatable setting.

The only known resizable tree-based ORAM construction is due to Moataz et al. [MMBC15]. However, this construction along with other tree-based ORAM constructions have logarithmic round complexity due to the need for recursively storing the position map in a smaller ORAM. Our StE construction avoids the need for a position map altogether, making our query and update protocols constant round.

2 Preliminaries

Basic terminology and notation. In this paper, *efficient* means *probabilistic polynomial-time* (in the input size). The security parameter will always be denoted k. We denote the empty string as ε. The symbol $\|$ denotes string concatenation. For a randomized algorithm $\mathcal{A}$, we write $y \xleftarrow{\$} \mathcal{A}(x)$ to denote running $\mathcal{A}$ on input x and letting y be a random variable representing its output.

Basic primitives. We will use CPA-secure symmetric encryption, pseudorandom functions (PRFs), and collision-resistant hash functions. As our theorems won't depend on the finer details of the definitions, we omit them. We refer the reader to, e.g., Katz and Lindell [KL20] for these definitions.

2.1 Two-Party Computation Definitions

Our treatment of *two-party protocols* is agnostic to details of how they are formally defined. We will consider *stateful* protocols where both parties accept inputs as well as some (possibly empty) previous state, and emit local outputs and some updated state. (This state refers to information saved *between* runs of the protocol, and not the information privately held by the parties *during* a run of the protocol.) When Π is a stateful two-party protocol, we write

$$(\mathsf{out}_1, \mathsf{st}_1;\ \mathsf{out}_2, \mathsf{st}_2 | \mathsf{V}_1, \mathsf{V}_2) \xleftarrow{\$} \Pi(\mathsf{in}_1, \mathsf{st}_1; \mathsf{in}_2, \mathsf{st}_2)$$

to denote running Π where party i gets input in_i and state input st_i, and emits output out_i and an updated state, and has *view* V_i (consisting of its random tape and all incoming messages). We

[4][KM22] supports a limited form of server-side querying and is designed for a weaker adversarial model than the one considered in our work.

Functionality $\mathcal{F}_{\mathsf{UPSI}}(\mathsf{in}_1, \mathsf{in}_2, \mathsf{st})$	Functionality $\mathcal{F}_{\mathsf{PSU}}(\mathsf{in}_1, \mathsf{in}_2)$
1 Parse $(X, Y) \leftarrow \mathsf{st}$	1 Parse $X \leftarrow \mathsf{in}_1; Y \leftarrow \mathsf{in}_2$
2 Parse $(X_+, X_-) \leftarrow \mathsf{in}_1, (Y_+, Y_-) \leftarrow \mathsf{in}_2$	2 Return $(X \cup Y, X \cup Y)$
3 $X_+ \leftarrow X_+ \setminus (X_- \cup X);\ X_- \leftarrow X_- \cap X$	
4 $Y_+ \leftarrow Y_+ \setminus (Y_- \cup Y);\ Y_- \leftarrow Y_- \cap Y$	
5 $X \leftarrow (X \setminus X_-) \cup X_+$	
6 $Y \leftarrow (Y \setminus Y_-) \cup Y_+$	
7 $\mathsf{st} \leftarrow (X, Y)$	
8 $(\mathsf{out}_1, \mathsf{out}_2) \leftarrow (X \cap Y, X \cap Y)$	
9 Return $(\mathsf{out}_1, \mathsf{out}_2, \mathsf{st})$	

Figure 1: Updatable set intersection functionality $\mathcal{F}_{\mathsf{UPSI}}$ and private set union functionality $\mathcal{F}_{\mathsf{PSU}}$. In both functionalities, if any input is not the prescribed form, the functionality returns $(\perp, \perp, \perp)$.

will also consider *stateless* (i.e., one-time) protocols, where we omit the state inputs and outputs. When they are not relevant, we omit the $\mathsf{V}_1, \mathsf{V}_2$ outputs from the notation.

A (deterministic) *two party reactive functionality* is, formally, any function $\mathcal{F} : (\{0,1\}^* \cup \{\perp\})^3 \to (\{0,1\}^* \cup \{\perp\})^3$. Following our emphasis on the full leakage profile of two-party protocols, we do not allow functionalities to be "partial"; they must be total functions (e.g. they must explicitly return errors if their input is not of the expected form). We also do not consider randomized functionalities in this paper. We will usually write the evaluation of a functionality $\mathcal{F}$ as $(\mathsf{out}_1, \mathsf{out}_2, \mathsf{st}_{\mathcal{F}}) \leftarrow \mathcal{F}(\mathsf{in}_1, \mathsf{in}_2, \mathsf{st}_{\mathcal{F}})$; Intuitively, the first two inputs to $\mathcal{F}$ correspond to the parties' inputs, and the third input the state of the functionality. The functionality outputs the parties' local outputs and an updated state.

We also define (non-reactive, deterministic) *functionalities* to be functions of the form $\mathcal{F} : (\{0,1\}^* \cup \{\perp\})^2 \to (\{0,1\}^* \cup \{\perp\})^2$. These can be interpreted similarly to the above definition, except that they do not have state inputs or outputs.

Figure 1 defines the reactive functionality $\mathcal{F}_{\mathsf{UPSI}}$ updateable private set intersection and the non-reactive functionality $\mathcal{F}_{\mathsf{PSU}}$ for private set union. In contrast to prior work, our version of $\mathcal{F}_{\mathsf{UPSI}}$ allows for set updates to vary in size, and even be malformed (e.g. deleting an element that is not already present). Similarly, $\mathcal{F}_{\mathsf{PSU}}$ allows for the sets X, Y to be of any size (though when they are chosen by a poly-time adversary, X and Y must be written down explicitly, which effectively limits their size when working with the functionality in security definitions). Compared to prior work, these definitions are more general in allowing flexibility for the users, but necessitate modified definitions (presented next) to be achievable in some cases.

Security of two-party computation for reactive functionalities. The following definition captures secure two-party computation of a reactive or non-reactive functionality against a passive, non-adaptive adversary.

Our definition notably departs from standard two-party computation definitions (see, e.g., [Lin17]) in that it explicitly models the *leakage* of a protocol in the style of structured encryption. This appears as a *leakage profile* $\mathcal{L} = (\mathcal{L}_1, \mathcal{L}_2)$, a pair of of algorithms where $\mathcal{L}_i$ computes the information required for simulation for party i. Traditionally this leakage is expressed as a "parameter" of a functionality, but our protocols will involve non-trivial leakage that is more properly expressed this way.

In our definitions, the adversary is allowed several invocations of the protocol from the point of view of one party, each of which mutate the state of the parties. For technical reasons, we consider a version of this definition where the adversary is allowed to ask for several sequential runs of the protocol with "resets" in between them. In traditional definitions, standard hybrid arguments can show that a single execution is equivalent to several with resets. However, in our setting with leakage profiles, this will no longer be the case. That is, it may be possible that a protocol has

some non-trivial leakage that is not noticeable in a single run but shows up as correlations between several runs.

Definition 2.1. Let Π be a two-party protocol, let $\mathcal{F}$ be a two-party reactive functionality, and let $\mathcal{L} = (\mathcal{L}_1, \mathcal{L}_2)$ be a pair of deterministic algorithms. We say that Π is a *secure two-party protocol for $\mathcal{F}$ with respect to leakage $\mathcal{L}$*, or *$\mathcal{L}$-secure*, if for $i = 1, 2$ and all efficient $\mathcal{A}$ there exists an efficient $\mathcal{S}$ such that

$$\Pr[\mathbf{2pcReal}^{\mathcal{A}}_{\Pi,i}(1^k) = 1] - \Pr[\mathbf{2pcIdeal}^{\mathcal{A}}_{\mathcal{F},\mathcal{L},\mathcal{S},i}(1^k) = 1]$$

is a negligible function of k, where $\mathbf{2pcReal}^{\mathcal{A}}_{\Pi,i}$ and $\mathbf{2pcIdeal}^{\mathcal{A}}_{\mathcal{F},\mathcal{L},\mathcal{S},i}$ are defined in Figure 2.

We now explain the meaning of the games, starting with $\mathbf{2pcReal}^{\mathcal{A}}_{\Pi,i}$. This game starts with $\mathcal{A}$ choosing a sequence $\overrightarrow{\mathsf{in}}$, where each entry consists of either a pair of inputs for the parties or a special symbol reset. Intuitively, entries of this vector indicate either that $\mathcal{A}$ would like the protocol run on these inputs, or to have the parties' private states set to empty, effectively restarting their interaction from scratch. The game then processes this vector to product $\overrightarrow{\mathsf{out}}, \overrightarrow{\mathsf{V}}$ for $\mathcal{A}$. Each such entry is produced by running Π on the chosen inputs, using state $\mathsf{st}_1, \mathsf{st}_2$ that are maintained in the game. When a reset symbol is encountered, the game simply returns $\mathsf{st}_1, \mathsf{st}_2$ to their initial empty states.

The ideal game $\mathbf{2pcIdeal}^{\mathcal{A}}_{\mathcal{F},\mathcal{L},\mathcal{S},i}$ starts by initializing states $\mathsf{st}_{\mathcal{F}}, \mathsf{st}_{\mathcal{L}}, \mathsf{st}_{\mathcal{S}}$. When $\mathcal{A}$ provides $\overrightarrow{\mathsf{in}}$, the game produces the individual views by invoking the functionality $\mathcal{F}$, and then the appropriate leakage function (either $\mathcal{L}_1$ or $\mathcal{L}_2$), and finally runs the simulator $\mathcal{S}$ on the input and output of the party, and the output λ of the leakage function to produce the view. Each of $\mathcal{F}, \mathcal{L}, \mathcal{S}$ maintains its own state, which are updated on each run. We note that, crucially, the outputs in $\overrightarrow{\mathsf{out}}$ are chosen by the functionality and not the simulator. Reset symbols are now processed by resetting only the state of the functionality (but *not* the simulator or leakage profile). The leakage profile and simulator are however notified of a reset on lines 6 and 7, where they are allowed to update their state.

Simplifications for stateless $\mathcal{F}$. When $\mathcal{F}$ is non-reactive, the definition simplifies considerably. In the real game, we can omit the states $\mathsf{st}_1, \mathsf{st}_2$, as the protocol is "one-shot". This means that resets become meaningless, and we can assume they are not submitted. In the ideal game, we now omit the functionality state $\mathsf{st}_{\mathcal{F}}$, but (importantly) keep the leakage and simulator states $\mathsf{st}_{\mathcal{L}}, \mathsf{st}_{\mathcal{S}}$ so that they can correlate the simulated views, if required. We can similarly assume that resets are not submitted (as $\mathcal{L}, \mathcal{S}$ know that there is no state to reset).

A smaller detail is that on line 1 of $\textsc{NextV}_i$ of the ideal game, we can omit the $\mathsf{out}_1, \mathsf{out}_2$ inputs to $\mathcal{L}$, since it can compute these itself. (With a stateful $\mathcal{F}$ this might not the be the case, since $\mathcal{L}$ does not have access to $\mathsf{st}_{\mathcal{F}}$.) We also note that our definitions apply a form of *correctness* in that the adversary can test if the output value it receives is correct according to $\mathcal{F}$.

Reverses of protocols. We will sometimes take two-party protocols and swap the roles of the parties. Formally, we define the *reverse* of a protocol Π, denoted Π^r, to the protocol resulting from switching the roles of the parties (include who speaks first). It is trivial that if Π is a $\mathcal{L}$-secure protocol for $\mathcal{F}$, then its reverse Π^r is $\mathcal{L}^r$ secure for $\mathcal{F}^r$, where $\mathcal{L}^r$ and $\mathcal{F}^r$ interchange their inputs and outputs from $\mathcal{F}$ and $\mathcal{L}$ in the obvious way.

2.2 Structured Encryption Definitions

We use two notions of *dynamic structured encryption (StE)* in this paper. Both model a set data structure, where a client can add and delete elements from a set, and then issue (batch) membership queries on the current set. Our first notion (Definition 2.2) is the standard one, where the client

Games $\mathbf{2pcReal}^{\mathcal{A}}_{\Pi,i}(1^k)$,

1. $(\overrightarrow{\mathsf{in}}, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}(1^k)$
2. $\mathsf{st}_1, \mathsf{st}_2 \leftarrow \perp;\ \overrightarrow{\mathsf{out}}, \overrightarrow{\mathsf{V}} \leftarrow \varepsilon$
3. For $j = 1, \ldots, |\overrightarrow{\mathsf{in}}|$:
4.   If $\overrightarrow{\mathsf{in}}[j] = \mathsf{reset}$:
5. $\mathsf{st}_1, \mathsf{st}_2 \leftarrow \perp$
6.   Else:
7. $(\mathsf{in}_1, \mathsf{in}_2) \leftarrow \overrightarrow{\mathsf{in}}[j]$
8. $(\mathsf{out}, \mathsf{V}) \leftarrow \textsc{NextV}_i(\mathsf{in}_1, \mathsf{in}_2)$
9. $\overrightarrow{\mathsf{out}} \leftarrow \overrightarrow{\mathsf{out}} \| \mathsf{out};\ \overrightarrow{\mathsf{V}} \leftarrow \overrightarrow{\mathsf{V}} \| \mathsf{V}$
10. $b \xleftarrow{\$} \mathcal{A}(\overrightarrow{\mathsf{out}}, \overrightarrow{\mathsf{V}}, \mathsf{st}_{\mathcal{A}})$
11. Return b

$\textsc{NextV}_i(\mathsf{in}_1, \mathsf{in}_2, \mathsf{st}_1, \mathsf{st}_2)$

1. $(\mathsf{out}_1, \mathsf{st}_1; \mathsf{out}_2, \mathsf{st}_2 | \mathsf{V}_1, \mathsf{V}_2)$
    $\xleftarrow{\$} \Pi(\mathsf{in}_1, \mathsf{st}_1; \mathsf{in}_2, \mathsf{st}_2)$
2. Return $\mathsf{out}_i, \mathsf{V}_i$

Game $\mathbf{2pcIdeal}^{\mathcal{A}}_{\mathcal{F},\mathcal{L},\mathcal{S},i}(1^k)$

1. $(\overrightarrow{\mathsf{in}}, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}(1^k)$
2. $\mathsf{st}_{\mathcal{F}}, \mathsf{st}_{\mathcal{S}}, \mathsf{st}_{\mathcal{L}} \leftarrow \perp; \overrightarrow{\mathsf{out}}, \overrightarrow{\mathsf{V}} \leftarrow \varepsilon$
3. For $j = 1, \ldots, |\overrightarrow{\mathsf{in}}|$:
4.   If $\overrightarrow{\mathsf{in}}[j] = \mathsf{reset}$:
5. $\mathsf{st}_{\mathcal{F}} \leftarrow \perp$
6. $\mathsf{st}_{\mathcal{L}} \leftarrow \mathcal{L}(\mathsf{reset}, \mathsf{st}_{\mathcal{L}})$
7. $\mathsf{st}_{\mathcal{S}} \leftarrow \mathcal{S}(\mathsf{reset}, \mathsf{st}_{\mathcal{S}})$
8.   Else:
9. $(\mathsf{in}_1, \mathsf{in}_2) \leftarrow \overrightarrow{\mathsf{in}}[j]$
10. $(\mathsf{out}, \mathsf{V}) \leftarrow \textsc{NextV}_i(\mathsf{in}_1, \mathsf{in}_2)$
11. $\overrightarrow{\mathsf{out}} \leftarrow \overrightarrow{\mathsf{out}} \| \mathsf{out};\ \overrightarrow{\mathsf{V}} \leftarrow \overrightarrow{\mathsf{V}} \| \mathsf{V}$
12. $b \xleftarrow{\$} \mathcal{A}(\overrightarrow{\mathsf{out}}, \overrightarrow{\mathsf{V}}, \mathsf{st}_{\mathcal{A}})$
13. Return b

$\textsc{NextV}_i(\mathsf{in}_1, \mathsf{in}_2)$

1. $(\mathsf{out}_1, \mathsf{out}_2, \mathsf{st}_{\mathcal{F}}) \leftarrow \mathcal{F}(\mathsf{in}_1, \mathsf{in}_2, \mathsf{st}_{\mathcal{F}})$
2. $(\lambda, \mathsf{st}_{\mathcal{L}}) \xleftarrow{\$} \mathcal{L}_i(\mathsf{in}_1, \mathsf{in}_2, \mathsf{out}_1, \mathsf{out}_2, \mathsf{st}_{\mathcal{L}})$
3. $(\mathsf{V}, \mathsf{st}_{\mathcal{S}}) \xleftarrow{\$} \mathcal{S}(\mathsf{in}_i, \mathsf{out}_i, \lambda, \mathsf{st}_{\mathcal{S}})$
4. Return $\mathsf{out}_i, \mathsf{V}$

Figure 2: Games $\mathbf{2pcReal}^{\mathcal{A}}_{\Pi,i}$ and $\mathbf{2pcIdeal}^{\mathcal{A}}_{\mathcal{F},\mathcal{L},\mathcal{S},i}$ used in Definition 2.1.

issues queries, and follows works like [CK10]. Intuitively, security is only guaranteed for the client, as the server has no private inputs. We only use this definition for presenting our construction ESX for standalone purposes.

Our second notion (Definition 2.5) is new, and will be used in our general framework. This notion, which we call *StE with server-side querying*, allows the *server* to make queries instead of the client. The server's input is considered private, so the security definition includes conditions for both parties, in the style of two-party computation. Our approach will be to construct a standard StE scheme and then modify it (using standard tools) to support server-side querying.

Our security and correctness definitions for both types of StE will be with respect to *non-adaptive* adversaries who declare all of the parties' inputs up-front. This is because the notion of two-party computation we target for updatable PSI only requires these weaker definitions.

The next definitions introduce both types of StE.

Definition 2.2. A *dynamic structured encryption (StE) scheme for the set datatype* is a pair of two-party protocols $\Sigma = (\mathsf{Qry}, \mathsf{Upd})$ with the following syntax.

- Qry is a protocol where the first party (i.e. the client) accepts as input a state st and a set $X_{\text{qry}} \subseteq \{0,1\}^*$, and the second party (i.e. the server) accepts as input an encrypted set ES. The first party outputs a set $X_{\text{out}} \subseteq \{0,1\}^*$ and the second party has no output. We will usually denote this via
$$(X_{\text{out}}; \perp | \mathsf{V}_1, \mathsf{V}_2) \xleftarrow{\$} \mathsf{Qry}(\mathsf{st}, X_{\text{qry}}; \mathsf{ES}).$$
- Upd is a protocol where the first party inputs a state st and sets (X_+, X_-), and the second party inputs an encrypted set ES. The first party outputs an updated state st and second party outputs an updated ES. We will usually denote this via
$$(\mathsf{st}; \mathsf{ES} | \mathsf{V}_1, \mathsf{V}_2) \xleftarrow{\$} \mathsf{Upd}(\mathsf{st}, (X_+, X_-); \mathsf{ES}).$$

A *dynamic structured encryption (StE) scheme with server-side querying for the set datatype* is a pair of two protocols $\Sigma = (\mathsf{SQry}, \mathsf{Upd})$ where Upd has the same syntax above, and SQry has

Game $\mathbf{Cor}_{\Sigma}^{\mathcal{A}}(1^k)$	$\textsc{Upd}(X_+, X_-)$
1 $\overrightarrow{\mathsf{op}} \xleftarrow{\$} A(1^k)$	1 $(\mathsf{st}; \mathsf{ES}) \xleftarrow{\$} \mathsf{Upd}(\mathsf{st}, (X_+, X_-); \mathsf{ES})$
2 $X \leftarrow \emptyset$; $\textsc{Win} \leftarrow 0, \mathsf{st} \leftarrow \bot$	2 $X \leftarrow (X \setminus X_-) \cup X_+$
3 For $j = 1, \ldots, \overrightarrow{\mathsf{op}}$:	$\textsc{Qry}(X_{\text{qry}})$
4 If $(X_+, X_-) \leftarrow \overrightarrow{\mathsf{op}}[j]$: $\textsc{Upd}(X_+, X_-)$	1 $\boxed{(X_{\text{out}}; \bot) \xleftarrow{\$} \mathsf{Qry}(\mathsf{st}, X_{\text{qry}}; \mathsf{ES})}$
5 Else If $X_{\text{qry}} \leftarrow \overrightarrow{\mathsf{op}}[j]$: $\textsc{Qry}(X_{\text{qry}})$	2 $(\bot; X_{\text{out}}) \xleftarrow{\$} \mathsf{SQry}(\mathsf{st}; X_{\text{qry}}, \mathsf{ES})$
6 Return $\textsc{Win}$	3 If $X_{\text{out}} \neq X \cap X_{\text{qry}}$: $\textsc{Win} \leftarrow 1$

Figure 3: Game $\mathbf{Cor}_{\Sigma}^{\mathcal{A}}$ used in Definition 2.3. When Σ is a standard StE scheme, the game uses the boxed code. When Σ supports server-side querying, the shaded code is used instead.

similar syntax to Qry above, except that the server inputs X_{qry} and receives the output X_{out}, and the client receives no output. We usually denote this via

$$(\bot; X_{\text{out}} | \mathsf{V}_1, \mathsf{V}_2) \xleftarrow{\$} \mathsf{SQry}(\mathsf{st}; X_{\text{qry}}, \mathsf{ES}).$$

We next define correctness for both types of StE. We cannot use a simpler definition (e.g. where answers are correct with probability one) since constructions will typically err with negligible (but non-zero) probability, for example when hash collisions occur.

Definition 2.3. Let Σ be a dynamic StE scheme (with or without server-side querying) for the set datatype. We say that Σ is *correct* if for all efficient $\mathcal{A}$, $\Pr[\mathbf{Cor}_{\Sigma}^{\mathcal{A}}(1^k) = 1]$ is a negligible function in k, where $\mathbf{Cor}_{\Sigma}^{\mathcal{A}}$ is defined in Figure 3.

StE security. We next recall a standard non-adaptive real/ideal definition of security of traditional StE (without server-side querying) with respect to a leakage profile $\mathcal{L}$ [CK10]. This definition intuitively requires that whatever is learned by the server is limited to the output λ of $\mathcal{L}$.

Definition 2.4. Let $\Sigma = (\mathsf{Qry}, \mathsf{Upd})$ be an StE scheme for the set datatype and let $\mathcal{L}$ be an algorithm. We say that Σ is a *$\mathcal{L}$-secure against passive persistent attacks*, or simply *$\mathcal{L}$-secure*, if for all efficient $\mathcal{A}$ there exists an efficient $\mathcal{S}$ such that

$$\Pr[\mathbf{StE\text{-}Real}_{\Pi}^{\mathcal{A}}(1^k) = 1] - \Pr[\mathbf{StE\text{-}Ideal}_{\mathcal{L},\mathcal{S}}^{\mathcal{A}}(1^k) = 1]$$

is a negligible function of k, where $\mathbf{StE\text{-}Real}_{\Pi}^{\mathcal{A}}$ and $\mathbf{StE\text{-}Ideal}_{\mathcal{L},\mathcal{S}}^{\mathcal{A}}$ are defined in Figure 4.

We next define security for StE with server-side querying. As mentioned above, this requires defining security for both parties, since now updates should be private from the server and queries should be private from the client. In the following, we use the subscript $i = 1$ to denote security for the server's queries, and $i = 2$ to denote security for the client's updates.

Definition 2.5. Let $\Sigma = (\mathsf{SQry}, \mathsf{Upd})$ be an StE scheme with server-side querying for the set datatype and let $\mathcal{L} = (\mathcal{L}_1, \mathcal{L}_2)$ be a pair of algorithms. We say that Σ is a *$\mathcal{L}$-secure against passive persistent attacks*, or simply *$\mathcal{L}$-secure*, if for $i = 1, 2$ and all efficient $\mathcal{A}$ there exists an efficient $\mathcal{S}$ such that

$$\Pr[\mathbf{SQStE\text{-}Real}_{\Pi,i}^{\mathcal{A}}(1^k) = 1] - \Pr[\mathbf{SQStE\text{-}Ideal}_{\mathcal{L}_i,\mathcal{S},i}^{\mathcal{A}}(1^k) = 1]$$

is a negligible function of k, where $\mathbf{SQStE\text{-}Real}_{\Pi,i}^{\mathcal{A}}$ and $\mathbf{SQStE\text{-}Ideal}_{\mathcal{L}_i,\mathcal{S},i}^{\mathcal{A}}$ are defined in Figure 5.

We draw attention to some details in the ideal game used to define security. In the case $i = 1$, where $\mathcal{S}$ is simulating the client's view, $\mathcal{S}$ is given inputs X_+, X_- during updates but is not given X_{qry} during queries. This represents that the client knows its own inputs, but does not know the server query. We need to give X_+, X_- to the leakage profile to allow it to update its state for future leakage computation. We made a similar choice in the $i = 2$ case (i.e. security for the client), where X_{qry} is now given to the simulator during queries, but X_+, X_- are not given to the simulator during updates.

Game $\textbf{StE-Real}_{\Pi}^{\mathcal{A}}(1^k)$

1 $(\overrightarrow{\mathsf{op}}, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} A(1^k)$
2 $\mathsf{st}, \mathsf{ES} \leftarrow \bot; \overrightarrow{\mathsf{V}} \leftarrow \varepsilon$
3 For $j = 1, \ldots, |\overrightarrow{\mathsf{op}}|$:
4   If $(X_+, X_-) \leftarrow \overrightarrow{\mathsf{op}}[j]$
5         $\mathsf{V} \leftarrow \textsc{Upd}(X_+, X_-)$
6   Else If $X_{\text{qry}} \leftarrow \overrightarrow{\mathsf{op}}[j]$
7         $\mathsf{V} \leftarrow \textsc{Qry}(X_{\text{qry}})$
8     $\overrightarrow{\mathsf{V}} \leftarrow \overrightarrow{\mathsf{V}} \| \mathsf{V}$
9 $b \xleftarrow{\$} A(\overrightarrow{\mathsf{V}}, \mathsf{st}_{\mathcal{A}})$
10 Return b

$\textsc{Qry}(X_{\text{qry}})$

1 $(\bot; X_{\text{out}} | \mathsf{V}_1, \mathsf{V}_2) \xleftarrow{\$} \mathsf{Qry}(\mathsf{st}, X_{\text{qry}}; \mathsf{ES})$
2 Return V_2

$\textsc{Upd}(X_+, X_-)$

1 $(\mathsf{st}; \mathsf{ES} | \mathsf{V}_1, \mathsf{V}_2) \xleftarrow{\$} \mathsf{Upd}(\mathsf{st}, (X_+, X_-); \mathsf{ES})$
2 Return V_2

Game $\textbf{StE-Ideal}_{\mathcal{L},\mathcal{S}}^{\mathcal{A}}(1^k)$

1 $(\overrightarrow{\mathsf{op}}, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} A(1^k)$
2 $\mathsf{st}, \mathsf{ES}, \mathsf{st}_{\mathcal{L}}, \mathsf{st}_{\mathcal{S}} \leftarrow \bot; \overrightarrow{\mathsf{V}} \leftarrow \varepsilon$
3 For $j = 1, \ldots, |\overrightarrow{\mathsf{op}}|$:
4   If $(X_+, X_-) \leftarrow \overrightarrow{\mathsf{op}}[j]$
5         $\mathsf{V} \leftarrow \textsc{Upd}(X_+, X_-)$
6   Else If $X_{\text{qry}} \leftarrow \overrightarrow{\mathsf{op}}[j]$
7         $\mathsf{V} \leftarrow \textsc{Qry}(X_{\text{qry}})$
8     $\overrightarrow{\mathsf{V}} \leftarrow \overrightarrow{\mathsf{V}} \| \mathsf{V}$
9 $b \xleftarrow{\$} A(\overrightarrow{\mathsf{V}}, \mathsf{st}_{\mathcal{A}})$
10 Return b

$\textsc{Qry}(X_{\text{qry}})$

1 $(\lambda, \mathsf{st}_{\mathcal{L}}) \xleftarrow{\$} \mathcal{L}(X_{\text{qry}}, \mathsf{st}_{\mathcal{L}})$
2 $(\mathsf{V}, \mathsf{st}_{\mathcal{S}}) \xleftarrow{\$} \mathcal{S}(\lambda, \mathsf{st}_{\mathcal{S}})$
3 Return V

$\textsc{Upd}(X_+, X_-)$

1 $(\lambda, \mathsf{st}_{\mathcal{L}}) \xleftarrow{\$} \mathcal{L}(X_+, X_-, \mathsf{st}_{\mathcal{L}})$
2 $(\mathsf{V}, \mathsf{st}_{\mathcal{S}}) \xleftarrow{\$} \mathcal{S}(\lambda, \mathsf{st}_{\mathcal{S}})$
3 Return V

Figure 4: Games $\textbf{StE-Real}_{\Pi}^{\mathcal{A}}$, $\textbf{StE-Ideal}_{\mathcal{L},\mathcal{S}}^{\mathcal{A}}$ used in Definition 2.4.

Game $\textbf{SQStE-Real}_{\Pi,i}^{\mathcal{A}}(1^k)$ $(i = 1, 2)$

1 $(\overrightarrow{\mathsf{op}}, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} A(1^k)$
2 $\mathsf{st}, \mathsf{ES} \leftarrow \bot; \overrightarrow{\mathsf{V}} \leftarrow \varepsilon$
3 For $j = 1, \ldots, |\overrightarrow{\mathsf{op}}|$:
4   If $(X_+, X_-) \leftarrow \overrightarrow{\mathsf{op}}[j]$:
5         $\mathsf{V} \leftarrow \textsc{Upd}(X_+, X_-)$
6   Else If $X_{\text{qry}} \leftarrow \overrightarrow{\mathsf{op}}[j]$:
7         $\mathsf{V} \leftarrow \textsc{SQry}(X_{\text{qry}})$
8     $\overrightarrow{\mathsf{V}} \leftarrow \overrightarrow{\mathsf{V}} \| \mathsf{V}$
9 $b \xleftarrow{\$} A(\overrightarrow{\mathsf{V}}, \mathsf{st}_{\mathcal{A}})$
10 Return b

$\textsc{SQry}(X_{\text{qry}})$

1 $(\bot; X_{\text{out}} | \mathsf{V}_1, \mathsf{V}_2) \xleftarrow{\$} \mathsf{SQry}(\mathsf{st}; X_{\text{qry}}, \mathsf{ES})$
2 Return V_i

$\textsc{Upd}(X_+, X_-)$

1 $(\mathsf{st}; \mathsf{ES} | \mathsf{V}_1, \mathsf{V}_2) \xleftarrow{\$} \mathsf{Upd}(\mathsf{st}, (X_+, X_-); \mathsf{ES})$
2 Return V_i

Game $\textbf{SQStE-Ideal}_{\mathcal{L},\mathcal{S},i}^{\mathcal{A}}(1^k)$

1 $(\overrightarrow{\mathsf{op}}, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} A(1^k)$
2 $\mathsf{st}_{\mathcal{L}}, \mathsf{st}_{\mathcal{S}} \leftarrow \bot; \overrightarrow{\mathsf{V}} \leftarrow \varepsilon$
3 For $j = 1, \ldots, |\overrightarrow{\mathsf{op}}|$:
4   If $(X_+, X_-) \leftarrow \overrightarrow{\mathsf{op}}[j]$:
5         $\mathsf{V} \leftarrow \textsc{Upd}(X_+, X_-)$
6   Else If $X_{\text{qry}} \leftarrow \overrightarrow{\mathsf{op}}[j]$:
7         $\mathsf{V} \leftarrow \textsc{SQry}(X_{\text{qry}})$
8     $\overrightarrow{\mathsf{V}} \leftarrow \overrightarrow{\mathsf{V}} \| \mathsf{V}$
9 $b \xleftarrow{\$} A(\overrightarrow{\mathsf{V}}, \mathsf{st}_{\mathcal{A}})$
10 Return b

$\textsc{SQry}(X_{\text{qry}})$ $(i = 1)$

1 $(\lambda, \mathsf{st}_{\mathcal{L}}) \xleftarrow{\$} \mathcal{L}_1(X_{\text{qry}}, \mathsf{st}_{\mathcal{L}})$
2 $(\mathsf{V}, \mathsf{st}_{\mathcal{S}}) \xleftarrow{\$} \mathcal{S}(\lambda, \mathsf{st}_{\mathcal{S}})$
3 Return V

$\textsc{Upd}(X_+, X_-)$ $(i = 1)$

1 $(\lambda, \mathsf{st}_{\mathcal{L}}) \xleftarrow{\$} \mathcal{L}_1(X_+, X_-, \mathsf{st}_{\mathcal{L}})$
2 $(\mathsf{V}, \mathsf{st}_{\mathcal{S}}) \xleftarrow{\$} \mathcal{S}(X_+, X_-, \lambda, \mathsf{st}_{\mathcal{S}})$
3 Return V

$\textsc{SQry}(X_{\text{qry}})$ $(i = 2)$

1 $(\lambda, \mathsf{st}_{\mathcal{L}}) \xleftarrow{\$} \mathcal{L}_2(X_{\text{qry}}, \mathsf{st}_{\mathcal{L}})$
2 $(\mathsf{V}, \mathsf{st}_{\mathcal{S}}) \xleftarrow{\$} \mathcal{S}(X_{\text{qry}}, \lambda, \mathsf{st}_{\mathcal{S}})$
3 Return V

$\textsc{Upd}(X_+, X_-)$ $(i = 2)$

1 $(\lambda, \mathsf{st}_{\mathcal{L}}) \xleftarrow{\$} \mathcal{L}_2(X_+, X_-, \mathsf{st}_{\mathcal{L}})$
2 $(\mathsf{V}, \mathsf{st}_{\mathcal{S}}) \xleftarrow{\$} \mathcal{S}(\lambda, \mathsf{st}_{\mathcal{S}})$
3 Return V

Figure 5: Games $\textbf{SQStE-Real}_{\Pi,i}^{\mathcal{A}}$ and $\textbf{SQStE-Ideal}_{\mathcal{L},\mathcal{S},i}^{\mathcal{A}}$ $(i = 1, 2)$ used in Definition 2.5.

2.3 Private Set Union

As a building block in our framework, we will also assume a non-reactive private set union (PSU) functionality (presented in Figure 1). It takes as input two sets X, Y from two parties respectively, and it outputs $X \cup Y$ to both parties. A number of concretely efficient solutions for 2 party PSU have been developed based based on OT and some form of oblivious shuffling protocol including [KRTW19, GMR+21, JSZ+22, ZCL+23, BPSY23]. All these constructions leak the size of the input sets to both parties, i.e. $\mathcal{L}_1(X, Y) = |Y|$ and $\mathcal{L}_2(X, Y) = |X|$.

3 Updatable PSI from Dynamic Structured Encryption

In this section we describe a general updatable PSI protocol supporting arbitrary inserts and deletes from any dynamic StE Σ with server-side querying and any PSU protocol Π_{PSU}. We prove the security of the protocol with respect to a leakage profile that is derived from the underlying leakage profiles of Σ and Π_{PSU}. We emphasize that while the framework is general, the instantiation of the underlying protocols must be done carefully to preserve the overall security and efficiency of the resulting updatable PSI protocol. Looking ahead, in Section 4 we will carefully construct a dynamic StE scheme with server-side querying and use our framework to construct our final updatable PSI protocol with *minimal leakage*, i.e., only the size of the updates.

Framework: Outline. Our framework is presented in Figure 6. At a high level, each party uses Σ to update their own encrypted set (held by the other party), and to server-side query the other party's set. Both parties use the Π_{PSU} protocol to compute the elements that must be added or removed from the intersection. We denote the parties P_X and P_Y with input sets X and Y, and refer to them as the left or the right party, respectively. We assume that each party already holds an encrypted set representing the other party's (previous) set, and that each party knows the intersection of the (previous) sets – denoted as I_1, I_2, where $I_1 = I_2$. In the first epoch of the protocol, these encrypted sets as well as the intersection can be considered empty. The framework now shows how to incorporate each party's inserts and deletes X_+, X_- and Y_+, Y_-, and compute the updated intersection.

Framework: Update and server query. To begin, both parties ensure that their inputs are well-formed (e.g., only deleting elements if they are in the sets). In the first stage of the framework, the left party acts as the client and runs the update protocol from Σ to perform the updates X_+, X_- on ES_X, held by the right party. The right party then uses the server-side query of Σ to query the updated ES_X with its additions Y_+. The second stage is symmetric, with the roles reversed. At the conclusion of the first and second stages the left and right parties receive sets S_1, S_2 respectively, which together consist of the elements that must be added to the intersection.[5]

Framework: PSU. Next, the parties run Π_{PSU} on the sets S_1, S_2 to learn their union (expressed as U_1, U_2, where $U_1 = U_2 = S_1 \cup S_2$). Next, the parties must compute the elements that must be removed from the current intersection. These elements are exactly the elements of the previous intersection that were deleted by one or both parties. In order to compute this, the parties run Π_{PSU} with the inputs $X_- \cap I_1$ and $Y_- \cap I_2$. The union of these sets (expressed as W_1, W_2) must be removed from the previous intersection.

Framework: Output. Finally, each party locally updates the previous intersection to compute the updated intersection.

[5] Recall that S_1 consists of elements that the left party added that are present in the updated set of the right party, and vice versa.

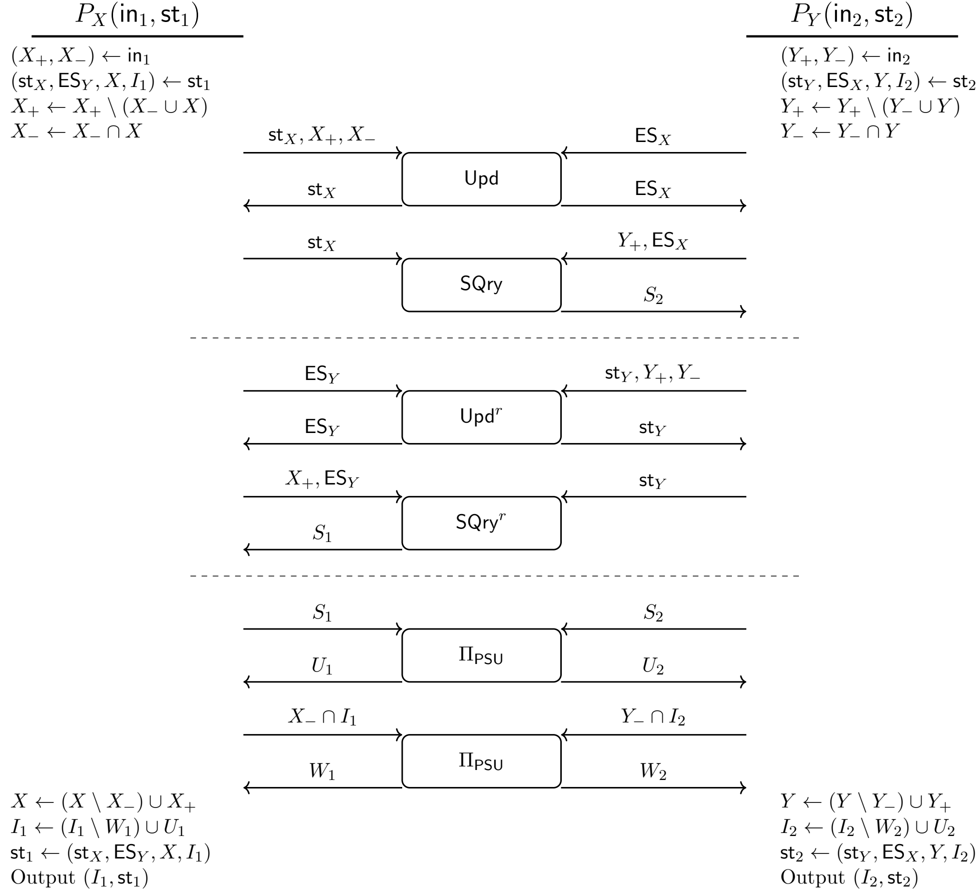

Figure 6: Our UPSI protocol Ω_{UPSI}.

$\mathcal{L}_1(\mathsf{in}_1, \mathsf{in}_2, \mathsf{out}_1, \mathsf{out}_2, \mathsf{st}_\mathcal{L})$

1. $(X, Y, \mathsf{st}_{\mathcal{L}_1^\Sigma}, \mathsf{st}_{\mathcal{L}_2^\Sigma}) \leftarrow \mathsf{st}_\mathcal{L}$
2. $(X_+, X_-) \leftarrow \mathsf{in}_1$; $(Y_+, Y_-) \leftarrow \mathsf{in}_2$
3. $I_{\text{prev}} \leftarrow X \cap Y$
4. $X \leftarrow (X \cup X_+) \setminus X_-$
5. $Y \leftarrow (Y \cup Y_+) \setminus Y_-$
6. $I_{\text{cur}} \leftarrow X \cap Y$
7. $S_1 \leftarrow I_{\text{cur}} \cap X_+$; $S_2 \leftarrow X \cap Y_+$
8. $R_1 \leftarrow I_{\text{prev}} \cap X_-$; $R_2 \leftarrow I_{\text{prev}} \cap Y_-$
9. $(\lambda^1, \mathsf{st}_{\mathcal{L}_1^\Sigma}) \xleftarrow{\$} \mathcal{L}_1^\Sigma(X_+, X_-, \mathsf{st}_{\mathcal{L}_1^\Sigma})$
10. $(\lambda^2, \mathsf{st}_{\mathcal{L}_1^\Sigma}) \xleftarrow{\$} \mathcal{L}_1^\Sigma(Y_+, \mathsf{st}_{\mathcal{L}_1^\Sigma})$
11. $(\lambda^3, \mathsf{st}_{\mathcal{L}_2^\Sigma}) \xleftarrow{\$} \mathcal{L}_2^\Sigma(Y_+, Y_-, \mathsf{st}_{\mathcal{L}_2^\Sigma})$
12. $(\lambda^4, \mathsf{st}_{\mathcal{L}_2^\Sigma}) \xleftarrow{\$} \mathcal{L}_2^\Sigma(X_+, \mathsf{st}_{\mathcal{L}_2^\Sigma})$
13. $\lambda^5 \xleftarrow{\$} \mathcal{L}_1^{\Pi_{\mathsf{PSU}}}(S_1, S_2)$
14. $\lambda^6 \xleftarrow{\$} \mathcal{L}_1^{\Pi_{\mathsf{PSU}}}(R_1, R_2)$
15. $\mathsf{st}_\mathcal{L} \leftarrow (X, Y, \mathsf{st}_{\mathcal{L}_1^\Sigma}, \mathsf{st}_{\mathcal{L}_2^\Sigma})$
16. Output $((\lambda^j)_{j=1}^6, \mathsf{st}_\mathcal{L})$

$\mathcal{S}_1(\mathsf{in}_1, \mathsf{out}_1, \lambda, \mathsf{st}_\mathcal{S})$

1. $(I_{\text{prev}}, \mathsf{st}_{\mathcal{S}_1^\Sigma}, \mathsf{st}_{\mathcal{S}_2^\Sigma}) \leftarrow \mathsf{st}_\mathcal{S}$
2. $(\lambda_j)_{j=1}^6 \leftarrow \lambda$
3. $(X_+, X_-) \leftarrow \mathsf{in}_1$; $I_{\text{cur}} \leftarrow \mathsf{out}_1$
4. $S_1 \leftarrow I_{\text{cur}} \cap X_+$
5. $U \leftarrow I_{\text{cur}} \setminus I_{\text{prev}}$
6. $R_1 \leftarrow X_- \cap I_{\text{prev}}$
7. $W \leftarrow I_{\text{prev}} \setminus I_{\text{cur}}$
8. $(\mathsf{V}_1^1, \mathsf{st}_{\mathcal{S}_1^\Sigma}) \xleftarrow{\$} \mathcal{S}_1^\Sigma((X_+, X_-), \lambda^1, \mathsf{st}_{\mathcal{S}_1^\Sigma})$
9. $(\mathsf{V}_1^2, \mathsf{st}_{\mathcal{S}_1^\Sigma}) \xleftarrow{\$} \mathcal{S}_1^\Sigma(\lambda^2, \mathsf{st}_{\mathcal{S}_1^\Sigma})$
10. $(\mathsf{V}_1^3, \mathsf{st}_{\mathcal{S}_2^\Sigma}) \xleftarrow{\$} \mathcal{S}_2^\Sigma(\lambda^3, \mathsf{st}_{\mathcal{S}_2^\Sigma})$
11. $(\mathsf{V}_1^4, \mathsf{st}_{\mathcal{S}_2^\Sigma}) \xleftarrow{\$} \mathcal{S}_2^\Sigma(X_+, \lambda^4, \mathsf{st}_{\mathcal{S}_2^\Sigma})$
12. $\mathsf{V}_1^5 \xleftarrow{\$} \mathcal{S}_1^{\Pi_{\mathsf{PSU}}}(S_1, U, \lambda^5)$
13. $\mathsf{V}_1^6 \xleftarrow{\$} \mathcal{S}_1^{\Pi_{\mathsf{PSU}}}(R_1, W, \lambda^6)$
14. $\mathsf{st}_\mathcal{S} \leftarrow (I_{\text{cur}}, \mathsf{st}_{\mathcal{S}_1^\Sigma}, \mathsf{st}_{\mathcal{S}_2^\Sigma})$
15. Output $((\mathsf{V}_1^j)_{j=1}^6, \mathsf{st}_\mathcal{S})$

Figure 7: Leakage profile and simulator for Theorem 3.1.

Security and leakage. We now state the formal security theorem for our framework and defer the proof to Appendix A.

Theorem 3.1. Suppose that $\Sigma = (\mathsf{SQry}, \mathsf{Upd})$ is a dynamic StE scheme with server-side querying for the set data type that is $\mathcal{L}^{\Sigma} = (\mathcal{L}_1^{\Sigma}, \mathcal{L}_2^{\Sigma})$-secure, and that Π_{PSU} is a stateless secure two-party protocol for the functionality $\mathcal{F}_{\mathsf{PSU}}$ (defined in Figure 1) with respect to leakage $\mathcal{L}^{\Pi_{\mathsf{PSU}}}$.

Then the construction Ω_{UPSI} given in Figure 6 is a secure protocol for the functionality $\mathcal{F}_{\mathsf{UPSI}}$ with respect to the leakage profile $\mathcal{L} = (\mathcal{L}_1, \mathcal{L}_2)$, where $\mathcal{L}_1$ is given on the left side of Figure 7 and $\mathcal{L}_2$ is the symmetric version of $\mathcal{L}_1$.

We now explain the resulting leakage profile $\mathcal{L}_1$ (the case of $\mathcal{L}_2$ is symmetric) in our framework. Lines 1 – 8 represent bookkeeping by the leakage profile to remember the parties' sets X, Y, the new and previous intersection, and the values S_1, S_2 (representing the elements in the new intersection that were newly added by each party) and R_1, R_2 (representing the elements from the previous intersection that were deleted by each party). Lines 9 – 14 compute the actual leakage. Lines 9 and 10 describe what the left party learns about Y_+, the additions from the right party, which would typically be only $|Y_+|$. Then lines 11 and 12 describe what this party learns about Y_+, Y_- during the other party's update and query stage. Once again, this would typically be only $|Y_+|$ and $|Y_-|$, but could be more or less information depending on the leakage of the underlying scheme Σ.

Next, lines 13 and 14 compute leakage on the intermediate values. A natural choice may be to let the left party learn $|S_2|$ and $|R_2|$, but this should be done with the awareness that this is non-trivial leakage about the other party's input that is conceivably harmful in applications. We note that prior work [BMX22] addressed this issue by assuming that the party's input sets (X_+, Y_+) were bounded by some publicly-known size, and padded smaller sets to this size before running the PSU protocol.

From our security analysis, in order to design an updatable PSI protocol with *minimal leakage* we would need to construct a dynamic StE scheme with server-side querying with minimal leakage as well, where the update and server-side query protocols of the StE scheme leak nothing more than the size of the update/query sets, and this will be our focus in the next section.

4 A Dynamic Encrypted Set Construction: ESX

In this section we construct an StE scheme for the set datatype that is compatible with our framework. We approach this by first building a construction that we call ESX that supports client-side querying in Subsection 4.1, which may be of independent interest. We construct ESX using symmetric-key primitives and it has only "query equality" leakage i.e. the server learns which client queries match across different query calls. For both update and query operations, our construction takes constant rounds, and both parties perform work that is polylogarithmic in the size of the accumulated set.

In Subsection 4.2 we show how to modify this ESX construction into one that supports server-side querying using some standard cryptographic protocols: Oblivious PRF and generic two-party computation. Most importantly, we show that this server-side querying StE has minimal leakage while it has the same asymptotic complexity as ESX.

4.1 Client-Side Querying Version

Our construction, ESX, is given in Figure 8 which protocols Qry and Upd along with routines Evict, ProcDels, WrtPth, and WrtBkt which are used by Upd.

Notation. These protocols perform operations on binary trees, which are not assumed to be complete. We implicitly assume that children of nodes are labeled as left or right. Given a bitstring $y \in \{0,1\}^*$ and a tree T, we write $\mathsf{Path}(\mathsf{T}, y)$ for the path that chooses the left or right child at each step according to the bits of y until it reaches a leaf. If y is longer than this path is deep, the remaining bits are ignored. In this path, we still assume that the children are labeled left or right. We similarly refer to the "node at y" (which may be undefined if y is too long). Unions of paths (such as the union on line 8 of Qry) will construct (non-full) binary trees. We write $|\mathsf{Path}(\mathsf{T}, y)|$ for the number of nodes on the path $\mathsf{Path}(\mathsf{T}, y)$.

ESX Construction: Data layout. As with all tree-based ORAMs, the server will maintain an encrypted binary tree, and each node of the tree will hold a bucket of some number of "slots" which may be either real or padding. In our construction, however, the size of the tree will gradually grow or shrink over time in *epochs*. We define an "epoch ending" to be when line 21 of Upd evaluates to true. During one epoch, the tree will either grow a new level (adding two leaves per operation), shrink one level (removing one leaf per operation), or stay the same. Also during an epoch, the size of each bucket can grow, shrink, or stay the same.

The decision to grow the tree and buckets is visible to the server and thus may leak information. We cannot simply track the number of real slots used in the tree and use this count, because whether or not deletions have been cleaned depends on the actual operations. Instead, to limit leakage, the decision to grow or shrink is determined by a *simulated load*, which is a pessimistic upper bound of the number of non-padding items (representing additions and deletions) in the tree at the end of an epoch. We insure that this upper bound depends *only* on the number of additions and deletions performed during each update, and can thus be simulated from a leakage profile that provides these.

ESX Construction: Ingredients. We use a standard CPA-secure encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ with k-bit keys. We abuse notation and feed trees to these algorithms to mean running encryption or decryption on every slot in a tree. We also use a variable-input-length PRF F that takes a k-bit key and produces a k-bit output.

The construction uses a routine $\mathsf{binrev}(k, t)$ that takes an positive integer k and an integer $0 \le t < 2^k$. It computes the standard k-bit representation of t and reverses it.

Finally the construction uses a padding $\mathsf{pad}(k, \mathsf{T}')$. On input positive integers k and a (partial) binary tree T', it pads all of the buckets in T' with plaintext dummy slots to some fixed size that depends on k. The theorem will specify our requirements on this operation.

ESX Construction: Querying. The query protocol is given on the left side of Figure 8. The client starts by initializing its state if this is the first time it has run: The routine $\mathsf{InitSt}(k)$ chooses $K_F, K_E, K_H \stackrel{\$}{\leftarrow} \{0,1\}^k$, sets $X \leftarrow \emptyset$, and sets $t_{\mathrm{ep}}, \mathsf{sLoad}, \mathsf{Dels}, \mathsf{curDels}$ all to zero. The client takes hashes of each element in its input set X_{qry}, applies the PRF F to the hashes, and sends the set of outputs τ to the server. By our assumption on F, the outputs are all k bits long.

Next, for each string $y \in \tau$ the server looks up the path in its tree T_c using y. Since y is k bits long, these paths will extend to a leaf of the tree (for sufficiently large values of the security parameter k; since the adversary is polynomial-time in k, the tree can be assumed to have depth at most k). The server forms a subtree T_c' of T_c as the union of these paths and send it to the client.

The client computes its output by decrypting the subtree T_c' to T'. For each element x of its input, it checks if (the hash of) x is present on its corresponding path an odd number of times, keeping it for the output if it is.

ESX Construction: Updating. The update protocol is given on the right side of Figure 8, with associate routines given lower in the figure. The client begins the protocol by initializing its state if necessary, and then lines 3–5 ensure that X_+, X_- have the appropriate form and update the

local copy of X. The client then simply tells the server total number of elements being added and deleted.

The server computes the next $n = |X_+| + |X_-|$ deterministic paths in this tree as determined by its state counter t, and let T'_c be their union and sends it to the client. The client decrypts T'_c to T and it removes the padding slots. Then for each element x to be added or deleted, it appends $H_{K_H}(x)$ to the root of T'_c and calls Evict on the next deterministic path; We comment below on how eviction works.

After eviction, it then checks if the current epoch has ended. The scheme maintains an invariant that at the end of each epoch the server's tree T_c is a complete binary tree with 2^h leaves, and at this point the client decides if the next epoch should add a level to the tree, remove a level, or leave the depth unchanged.

This decision is made according to sLoad, the "simulated load" on the tree, which is updated at the end of each epoch as follows: The client pessimistically assumes that t_{ep} new data items have been added to the tree (i.e. no delete operations were cleaned up), so it adds t_{ep} to sLoad. But we also know that the previous epoch's delete operations were cleaned up, so it subtracts $2 \cdot \mathsf{Dels}$. The client then adjusts h, the new tree height, using the updated sLoad on lines 26–27.

Finally, the client pads the tree T' using $\mathsf{pad}(k, \mathsf{T}')$, which adds extra padding slots to nodes of T'. Here, the nodes may grow or shrink in response to a change in sLoad. The client then encrypts T' and sends the result to the server, which overwrites the corresponding portion of its encrypted tree (including appending new nodes or deleting nodes, according to the T'_c that it receives).

ESX CONSTRUCTION: EVICTION. An eviction operation is called using a string y that specifies a path to leaf in T', along with a target height h. It starts by emptying the path into a multiset S and calling $\mathsf{ProcDels}(S)$. After this call, all items in S appear exactly one or zero times. It then checks the target height h and compares to the length of $\mathsf{Path}(\mathsf{T}', y)$. If the path is too short, then it adds two leaves, and if it is too long, it deletes the final leaf. It then calls WrtPth, which calls WrtBkt on each node of the path (plus possibly the two new leaves), and WrtBkt packs every item from S into a bucket if the path determined by the PRF goes through that bucket.

SECURITY. We now analyze our construction as an StE scheme. Intuitively, updates will leak only the number of additions and deletions in each, as the size of the tree (and all of its slots and buckets) can be inferred from these values alone. Queries will leakage an equality pattern because if x is queried multiple times, then the same leaf will be requested multiple times.

To express this formally, for a tuple qrys of n sets and an element x, define the *membership equality pattern* $\mathsf{meq}(\mathsf{qrys}, x) \in \{0,1\}^n$ with i-th bit indicating if x is a member of the i-th set of qrys.

Theorem 4.1. Let ESX be the construction given in Figure 8. Assume that $(\mathsf{Enc}, \mathsf{Dec})$ is CPA secure, that F is a secure PRF, and that H is collision-resistant. Assume the function $\mathsf{pad}(k, \mathsf{T})$ pads the nodes of T up to $\omega(k)$ slots. Then ESX is $\mathcal{L}_{\mathsf{ESX}}$-secure, where $\mathcal{L}_{\mathsf{ESX}}$ is the leakage profile is as follows:

Protocol $\mathsf{Qry}(\mathsf{st}, X_{\mathrm{qry}}; \mathsf{ES})$

Client:

1 If $\mathsf{st} = \bot$: $\mathsf{st} \leftarrow \mathsf{InitSt}(k)$
2 Parse st
3 $\tau \leftarrow \emptyset$
4 For $x \in X_{\mathrm{qry}}$: $\tau \leftarrow \tau \cup \{F_{K_F}(H_{K_H}(x))\}$
5 Send τ

Server:

6 If $\mathsf{ES} = \bot$: $\mathsf{ES} \leftarrow (\emptyset, 0)$
7 $(\mathsf{T}_c, t) \leftarrow \mathsf{ES}$
8 $\mathsf{T}'_c \leftarrow \emptyset$
9 For $y \in \tau$: $\mathsf{T}'_c \leftarrow \mathsf{T}'_c \cup \mathsf{Path}(\mathsf{T}_c, y)$
10 Send T'_c

Client:

11 $\mathsf{out} \leftarrow \emptyset$
12 $\mathsf{T}' \leftarrow \mathsf{Dec}(K_E, \mathsf{T}'_c)$
13 For $x \in X_{\mathrm{qry}}$:
14 $\quad y \leftarrow F_{K_F}(H_{K_H}(x))$
15 $\quad (B_1, \ldots, B_{h'}) \leftarrow \mathsf{Path}(\mathsf{T}', y)$
16 $\quad d \leftarrow 0$
17 $\quad$ For $i = 1, \ldots, h'$:
18 $\quad\quad$ If $H_{K_H}(x) \in B_i$: $d \leftarrow d + 1$
19 $\quad$ If d is odd: $\mathsf{out} \leftarrow \mathsf{out} \cup \{x\}$
20 Output out

Routine $\mathsf{ProcDels}(S)$

1 For $z \in S$:
2 $\quad$ If $\mathsf{mult}(z, S)$ even:
3 $\quad\quad$ Remove all z from S
4 $\quad$ Else:
5 $\quad\quad$ Remove all but one z from S
6 Return S

Routine $\mathsf{WrtPth}(K_F, \mathsf{T}', S, y, h)$

1 If $h = |y| + 1$:
2 $\quad (S, \mathsf{T}') \leftarrow \mathsf{WrtBkt}(K_F, \mathsf{T}', S, y_{1:h}\|0)$
3 $\quad (S, \mathsf{T}') \leftarrow \mathsf{WrtBkt}(K_F, \mathsf{T}', S, y_{1:h}\|1)$
4 For $i = h, \ldots, 0$:
5 $\quad (S, \mathsf{T}') \leftarrow \mathsf{WrtBkt}(K_F, \mathsf{T}', S, y_{1:i})$
6 return T'

Routine $\mathsf{WrtBkt}(K_F, \mathsf{T}', S, y')$

1 For $z \in S$:
2 $\quad$ If $F(K_F, z)_{1:|y'|} = y'$:
3 $\quad\quad$ Write z into node at y' in T'
4 $\quad\quad S \leftarrow S \setminus \{z\}$
5 return (S, T')

Protocol $\mathsf{Upd}(\mathsf{st}, (X_+, X_-); \mathsf{ES})$

Client:

1 If $\mathsf{st} = \bot$: $\mathsf{st} \leftarrow \mathsf{InitSt}(k)$
2 Parse st
3 $X_+ \leftarrow X_+ \setminus (X_- \cup X)$
4 $X_- \leftarrow X_- \cap X$
5 $X \leftarrow (X \setminus X_-) \cup X_+$
6 $n \leftarrow |X_+| + |X_-|$
7 Send n

Server:

8 $(\mathsf{T}_c, t) \leftarrow \mathsf{ES}$; $\mathsf{T}'_c \leftarrow \emptyset$
9 Repeat n times:
10 $\quad \mathsf{T}'_c \leftarrow \mathsf{T}'_c \cup \mathsf{Path}(\mathsf{T}_c, \mathsf{binrev}(k, t))$
11 $\quad t \leftarrow t + 1$
12 Send T'_c

Client:

13 $\mathsf{T}' \leftarrow \mathsf{Dec}(K_E, \mathsf{T}'_c)$; $\mathsf{T}' \leftarrow \mathsf{unpad}(\mathsf{T}')$
14 For each $x \in X_+$ and then for each $x \in X_-$:
15 $\quad$ If $x \in X_-$: $\mathsf{curDels} \leftarrow \mathsf{curDels} + 1$
16 $\quad$ Append $H_{K_H}(x)$ to the root of T'
17 $\quad$ Repeat 8 times:
18 $\quad\quad t_{\mathrm{ep}} \leftarrow t_{\mathrm{ep}} + 1$
19 $\quad\quad y \leftarrow \mathsf{binrev}(k, t); t \leftarrow t + 1$
20 $\quad\quad \mathsf{T}' \leftarrow \mathsf{Evict}(K_F, \mathsf{T}', y, h)$
21 $\quad\quad$ If $t_{\mathrm{ep}} = 2^h/8$: // *Epoch is over*
22 $\quad\quad\quad$ If $\mathsf{sLoad} < 2^h/8$: $h \leftarrow h - 1$
23 $\quad\quad\quad$ If $\mathsf{sLoad} \geq 2^h/4$: $h \leftarrow h + 1$
24 $\quad\quad\quad \mathsf{sLoad} = \mathsf{sLoad} - 2 \cdot \mathsf{Dels} + t_{\mathrm{ep}}$
25 $\quad\quad\quad \mathsf{Dels} \leftarrow \mathsf{curDels}$
26 $\quad\quad\quad \mathsf{curDels} \leftarrow 0$
27 $\quad\quad\quad t_{\mathrm{ep}} \leftarrow 0$
28 $\mathsf{T}' \leftarrow \mathsf{pad}(k, \mathsf{T}')$
29 $\mathsf{T}'_c \leftarrow \mathsf{Enc}(K_E, \mathsf{T}')$
30 Update st
31 Send T'_c; Output st

Server:

32 Write T'_c into T_c
33 $\mathsf{ES} \leftarrow (\mathsf{T}_c, t)$
34 Output ES

Routine $\mathsf{Evict}(K_F, \mathsf{T}', y, h)$

1 Empty nodes on $\mathsf{Path}(\mathsf{T}', y)$ into S
2 $S \leftarrow \mathsf{ProcDels}(S)$
3 If $|\mathsf{Path}(\mathsf{T}', y)| = h - 1$:
4 $\quad$ // *extend path with new leaves*
5 $\quad$ Add node at $y_{1:(h-1)}\|0$ to T'
6 $\quad$ Add node at $y_{1:(h-1)}\|1$ to T'
7 Else If $|\mathsf{Path}(\mathsf{T}', y)| = h + 1$:
8 $\quad$ // *shrink path by one node*
9 $\quad$ Delete node at $y_{1:(h+1)}$ from T'
10 $P \leftarrow \mathsf{WrtPth}(K_F, \mathsf{T}', S, y, h)$
11 Return T'

Figure 8: Our ESX construction $\Omega = (\mathsf{Qry}, \mathsf{Upd})$. Protocol Upd uses routines $\mathsf{Evict}, \mathsf{ProcDels}, \mathsf{WrtPth}$, and WrtBkt. $\mathsf{InitSt}()$ samples the three keys K_F, K_E, and K_H, sets X as $\emptyset$, and initializes $t, t_{\mathrm{ep}}, h, \mathsf{sLoad}, \mathsf{Dels}, \mathsf{curDels}$, all to 0. Line 2 in both protocols means to unpack the state, which contains $(K_F, K_E, K_H, X, t, t_{\mathrm{ep}}, h, \mathsf{sLoad}, \mathsf{Dels}, \mathsf{curDels})$.

```
L_ESX(in, st_L)
1  (X, qrys) ← st_L
2  X+ ← X+ \ (X− ∪ X)
3  X− ← X− ∩ X
4  If (X+, X−) ← in:   // Update
5      λ ← (|X+|, |X−|)
6  Else If X_qry ← in:   // Query
7      λ ← [ ]  // empty multiset
8      For x ∈ X_qry:
9          l ← meq(qrys, x)
10         λ ← λ ∪ [l]  // multiset union
11 X ← (X ∪ X+) \ X−
12 Output (λ, st_L)
```

Proof. Let $\mathcal{A}$ be an efficient adversary. We must give an efficient simulator $\mathcal{S}$ satisfying definition 2.4 with leakage profile $\mathcal{L}_{\mathsf{ESX}}$. The majority of this proof follows from standard techniques, so we only sketch them, and focus on the novel portion dealing with the overflow probability.

Via easy reductions, we can assume that all evaluations of the hash function H emit unique outputs, and we can also replace all evaluations of F with random k-bit strings. In this version of the game, a simulator can use the leakage profile to simulate the server's view during Qry protocol executions, which consists of τ. It receives as input a multiset λ indicating how elements intersected with past queries, and from this infers the size of X_{qry}. It simulates τ by selecting $|X_{\text{qry}}|$ random strings, reusing past strings as indicated.

For updates, the server's view is the first message n (which is easy to simulate) and T'_c. It is easy to simulate n from the leakage λ in the case of updates. For T'_c we observe that it consists of a tree data structure filled with freshly-encrypted ciphertexts, each computed on a k-bit plaintext. By the security of the encryption scheme, these can be encryptions of a fixed k-bit string instead.

To complete the proof, we must argue that the simulator can calculate the shape of T'_c, and then show that overflows happen with negligible probability. We start with the former. Given the update leakage $|X_+|, |X_-|$, the simulator can simulate the client logic the determines h in the update protocol: It starts with $\mathsf{sLoad} = h = 0$ and tracks $\mathsf{curDels}, \mathsf{Dels}, t_{\text{ep}}$ mimicking the protocol, except that uses the size of X_+ and X_- to determine how these variables change rather than the actual sets. Then using sLoad and h, the simulator can determine the shape of T'_c.

For the overflow analysis, we adapt the proof of Gentry et al. [GGH+13]. We will show that for any efficient adversary $\mathcal{A}$, and at any time during the execution of the protocol, any particular bucket overflows with negligible probability. A union bound over the polynomial number of time steps and buckets give the asymptotic bound.

We start by proving three invariants about our construction that control the "load" of the tree, meaning the number of real items in the tree, relative to its height.

Lemma 4.2. The following invariants hold for our construction:

1. At the end of every epoch, the load on tree is at most sLoad.
2. At every step, the load on the tree is at most $2^{\min\{h_0, h_1\}}$, where h_0 and h_1 are the heights of the tree at the beginning and end of the epoch.
3. At the end of every epoch, the load on the tree is at most 2^{h-1}, where h is the height of tree at that time.

A formal proof of the above lemma is presented in Appendix B. In our overflow analysis, we next fix an adversary that requests a total t update operations, and fix any bucket B of the final tree

after the adversary halts. It will be sufficient to prove that B overflows with negligible probability, as a union bound over all times and buckets shows that any overflow happens with negligible probability.

Let X be a random variable representing the number of items stored in B at the end. Then we can write $X = \sum_{i=1}^{t} X_i$, where X_i is an indicator the event that the item written at time i is in B at the end of the execution. Due to our lazy deletes, the X_i are *not* the sum of i.i.d. 0/1 random variables (because a delete operation will inject an item with the same leaf as a previous operation). However, this dependence only helps our analysis since *the same item cannot exist twice in any bucket*. This is because we immediately "clean up" paths to remove duplicates in ProcDels, and in particular the same item will never be placed in the same bucket twice. In terms of our X_i, this means that they are dependent, but only in the sense that for some i, j, $X_i = 1$ implies $X_j = 0$ (and similar relationships when one item is repeatedly added and deleted several times).

Thus we can write $X = X'$, where $X' = \sum_{i=1}^{t'} X'_i$ is a sum of $t' \le t$ independent random variables indicating if the i-th unique item appears in B. We calculate the expectation of X then apply a relative-error Chernoff bound, which does not require knowing t', and obtain a concentration bound for the number of items in B.

We proceed with the expectation calculation. Consider an epoch with starting and ending heights h_0, h_1, and let d be the depth of bucket B (here and below, length of paths refers to number of edges). Let c the length of the shortest path from B to one of its leaf ancestors (so $d+c \in \{h_0, h_1\}$, and d or c may be zero). We now claim that

Lemma 4.3. $\mathbb{E}[X] \le 2$.

Proof. We handle the cases $c < 2$ and $c \ge 2$ separately. In the first case, we observe that an item can be in B only if that item's leaf passes through B. By the second invariant, there are at most $2^{\min\{h_0,h_1\}}$ items in total, and each of these has a path through B with probability 2^{-d}. Therefore the expectation is at most

$$2^{\min\{h_0,h_1\}} \cdot 2^{-d} \le 2^{c+d} \cdot 2^{-d} \le 2,$$

where the final step uses that $c < 2$.

Now assume instead that $c \ge 2$. Since there is path of this length at least 2 below B, this is not the first visit to the node. The previous visit occurred 2^d steps ago, and after that visit there was at least one node below B, since the length of paths changes by at most one on each visit. The visits pass through two the distinct children below B, say u (on the last visit) and v (on the current visit). Any item assigned to B before the previous visit would have been flushed to either u or v or deeper. Any item assigned to B and through v will be flushed on the final visit. Therefore the only items in B after the final visit must be items assigned through u in the last 2^d operations. Since each of these has a leaf passing through u with probability $2^{-(d+1)}$, the expectation is at most

$$2^d \cdot 2^{-(d+1)} = 1/2,$$

which completes the proof of the lemma. □

To complete the overflow analysis, we use our observation above that $X = X'$, where X' is a sum of independent random variables. By a Chernoff bound and the lemma showing $\mathbb{E}[X] \le 2$, we have for every $\delta > 0$ that

$$\Pr[X' \ge 4(1+\delta)] \le e^{-4\delta^2/(2+\delta)}.$$

For any $Z \ge 8$, setting $\delta = (Z-4)/4$ gives

$$\Pr[X' \ge Z] \le e^{(Z-4)^2/(Z+4)} \le e^{-Z/6}.$$

Functionality $\mathcal{F}_{\mathsf{clnt}}(K_E; z, P)$	Protocol $\mathsf{SQry}(\mathsf{st}; X_{\mathrm{qry}}, \mathsf{ES})$ (Server computation)
1 Parse P as a list of ciphertexts	1 If $\mathsf{ES} = \bot$: $\mathsf{ES} \leftarrow (\emptyset, 0, K_H)$
2 $d \leftarrow 0$	2 For $x \in X_{\mathrm{qry}}$:
3 For $C \in P$:	3 Initiate $(\bot; y) \xleftarrow{\$} \Pi_F(K_F; H(K_H, x))$
4 If $\mathsf{Dec}(K_E, C) = z$: $d \leftarrow d + 1$	4 $P \leftarrow \mathsf{Path}(\mathsf{T}_c, y)$
5 $\mathsf{out}_1 \leftarrow \bot; \mathsf{out}_2 \leftarrow (d \bmod 2)$	5 Initiate $(\bot; b) \xleftarrow{\$} \Pi_{\mathsf{clnt}}(K_E; H(K_H, x), P)$
6 Return $(\mathsf{out}_1, \mathsf{out}_2)$	6 If $b = 1$: $\mathsf{out} \leftarrow \mathsf{out} \cup \{x\}$
	7 Output out

Figure 9: On the left is functionality $\mathcal{F}_{\mathsf{clnt}}(K_E; z, P)$ which is evaluated by Π_{clnt} as part of SQry. On the right is the server computation for protocol SQry used in the version of ESX with server-side querying. The client simply participates in the protocols $\Pi_F, \Pi_{\mathsf{clnt}}$ using K_F and K_E from its state.

Thus the overflow probability will be negligible when pad sets the bucket size to be $\omega(k)$ (which will be negligible in k for one bucket and also large enough to absorb the polynomial factors from the union bound). $\square$

4.2 Server-Side Querying Version

We now describe how to convert ESX which has query equality leakage into a server-side querying StE with minimal leakage. The update protocol remains exactly the same, and we must only modify the query protocol. At a high level, the conversion is simple: We replace the client's evaluation of the PRF F with an oblivious PRF, and then we replace the client's computation in the latter part of Qry with a two-party protocol for determining which x appears an odd number of times in the appropriate paths. Key to this approach is that the server can learn the intermediate PRF outputs and select the path from the tree it holds for the second part; This means that the second protocol only needs an input that scales with $\log |X|$ rather than the entire set.

SQry SUB-PROTOCOLS. We assume two protocols $\Pi_F, \Pi_{\mathsf{clnt}}$ have been constructed. The first protocol evaluates the functionality $\mathcal{F}(K_F; z)$ that outputs $(\bot; F(K_F, z))$, i.e. provides the right party with the PRF output and the left party with no output. The second performs the client computation from Qry where it determines if a given value appears in an even or odd number of ciphertexts. We formalize this functionality as $\mathcal{F}_{\mathsf{clnt}}$ on the left side of Figure 9. We remark that, for simplicity, we have this protocol operate on a single path P instead of a subtree (as was the case in Qry). It is possible to consider a batched version that works on the subtree, but the added complexity did not seem to bring any advantages.

SQry CONSTRUCTION. We describe the protocol on the right side of Figure 9. The code in that figure only gives the server computation; The client only needs to participate in the sub-protocols Π_F and Π_{clnt}. The protocol works by iterating over the inputs in X_{qry}, evaluating the PRF on each and then feeding the resulting path into Π_{clnt}. The server can then compute its results (and the client has no output).

SQry SECURITY. The client view can be simulated given just the size of X_{qry} - which matches exactly the leakage due to $\mathcal{L}_{\mathsf{clnt}}$. The server view for each update invocation can be simulated given just the sizes of sets X_+, X_- - which is the leakage for the client-query variant. For each query invocation this protocol has no leakage. That's because the server view containing correlated tree paths can be simulated given just the server input set X_{qry} - as the corresponding ESX protocol's server view can be simulated given just the query equality leakage. We encapsulate this leakage profile more formally in the following theorem.

Theorem 4.4. Suppose that Π_F is a stateless secure two-party protocol for the functionality $\mathcal{F}$ with respect to leakage $\mathcal{L}_F$ that only leaks the parties' input lengths, and that Π_{clnt} is a stateless secure two-party protocol for the functionality $\mathcal{F}_{\mathsf{clnt}}$ with respect to leakage $\mathcal{L}_{\mathsf{clnt}}$ that also only

leaks . Further assume the same conditions as in Theorem 4.1. Then esx-sqry is $\mathcal{L}_{\textsf{esx-sqry}}$-secure, where $\mathcal{L}_{\textsf{esx-sqry}} = (\mathcal{L}_{\textsf{esx-sqry},1}, \mathcal{L}_{\textsf{esx-sqry},2})$ is the leakage profile is as follows:

$\mathcal{L}_{\textsf{esx-sqry},1}(\textsf{in})$	$\mathcal{L}_{\textsf{esx-sqry},2}(\textsf{in}, \textsf{st}_{\mathcal{L}})$
1 If $(X_+, X_-) \leftarrow \textsf{in}$: // *Update*	1 $X \leftarrow \textsf{st}_{\mathcal{L}}$
2 $\lambda \leftarrow \perp$	2 $X_+ \leftarrow X_+ \setminus (X_- \cup X)$
3 Else If $X_{\text{qry}} \leftarrow \textsf{in}$: // *Query*	3 $X_- \leftarrow X_- \cap X$
4 $\lambda \leftarrow \lvert X_{\text{qry}} \rvert$	4 If $(X_+, X_-) \leftarrow \textsf{in}$: // *Update*
5 $X \leftarrow (X \cup X_+) \setminus X_-$	5 $\lambda \leftarrow (\lvert X_+ \rvert, \lvert X_- \rvert)$
6 Output $(\lambda, \textsf{st}_{\mathcal{L}})$	6 Else If $X_{\text{qry}} \leftarrow \textsf{in}$: // *Query*
	7 $\lambda \leftarrow \perp$
	8 $\textsf{st}_{\mathcal{L}} \leftarrow (X \cup X_+) \setminus X_-$
	9 Output $(\lambda, \textsf{st}_{\mathcal{L}})$

The leakage to client is stateless and only leaks the size of X_{qry}, while the leakage to the server consists of the number of valid additions and deletions. We recall that in the case of updates for the client, the leakage consists of *extra* information beyond its input X_+, X_-, and similarly for queries for the server with X_{qry}.

Instantiation. One very simple, yet efficient way to construct Π_F and $\Pi_{\textsf{clnt}}$ is to use an Oblivious PRF protocol [FIPR05, JL09, APRR24] and generic 2PC based on garbled circuits [Yao86, LP09] respectively.

The OPRF protocol due to Alamati et. al [APRR24] based on OT and alternating-moduli PRF implements the functionality Π_F in 2 rounds, where the computation cost of both parties is $O(k)$ bit operations and the communication cost is $O(k)$ bits.

Garbled circuits can be used to implement a non-reactive functionality where both parties input (C, x) and (C', y) respectively. Further the functionality parses the two inputs C, C' as circuits in some canonical representation, and the first party outputs $C(x, y)$ if $C = C'$ and otherwise it outputs $\perp$. The output of the second party is $\perp$. Hence, note that the $\Pi_{\textsf{clnt}}$ functionality can be implemeted by garbled circuits functionality, where the code of $\Pi_{\textsf{clnt}}$ can be translated into a simple circuit implementing decryptions and counting modulo 2. The state of the art garbled circuit instantiation due to Rosulek and Roy [RR21] have communication cost of $1.5k$ per AND gate, and ≤ 6 calls to a circular correlation robust (CCR) hash function per AND gate for both parties.

Hence, the computation complexity of SQry of both parties is dominated by $\omega(|X_{\text{qry}}|k^2 \log |X|)$ CCR hashes and the communication cost is dominated by the size of the garbled circuit: $\omega(|X_{\text{qry}}|k^3 \log |X|)$.

5 Putting it all Together - Our Updatable PSI Protocol

We can now plug in our new ESX construction (from Section 4.2) with server side querying and the PSU protocol due to Zhang et al. [ZCL+23] or Bienstock et al. [BPSY23] into the updatable PSI framework from Section 3 to get an updatable PSI protocol. Our ESX with server side querying has minimum leakage. We can ensure that in each epoch, the PSU invocations only leak $|X_+|, |X_-|, |Y_+|, |Y_-|$ by padding the input sets $S_1, S_2, (X_- \cap I_1), (Y_- \cap I_2)$ in stage 3 of the protocol with dummy elements, so each of these input sets contain $|X_+|, |X_-|, |Y_+|, |Y_-|$ elements respectively. Hence, our updatable PSI protocol also has *minimal leakage* in each epoch - leaking nothing more than the size of the insert and delete sets.

Complexity Let $\eta_+ = |X_+| + |Y_+|$ and $\eta_- = |X_-| + |Y_-|$. Then asymptotic communication complexity in any epoch of our updatable PSI is $\omega(k \log(|X||Y|)(k^2\eta_+ + \eta_-))$. The computation complexity of the first party is dominated by $O(k(\log(|X||Y|)\ (k\eta_1 + \eta_2))$ hashes. The computation

cost of party P_Y can be similarly obtained (by just reversing the sets X in Y in the complexity of P_X). A more fine grained breakdown of the complexity per stage of the protocol is listed next:

- Stage 1: The communication cost of Upd invocation is $\omega(\log|X|k(|X_+|+|X_-|))$ and SQry invocation is $\omega(\log|Y|k^3|Y_+|)$. The computation cost of P_X is dominated by $O(\log|Y|k^2|Y_+|)$ hashes.
- Stage 2: The communication cost of Upd^r invocation is $\omega(\log|Y|k(|Y_+|+|Y_-|))$ and SQry^r invocation is $\omega(\log|X|k^3|X_+|)$. The computation cost of P_X is dominated by $O(\log|X|k^2|X_+|)$ hashes.
- Stage 3: The communication cost of the first and the second PSU invocation are $O(k\eta_+)$ and $O(k\eta_-)$ respectively. The computation cost of both parties is dominated by the $O(\eta_+ + \eta_-)$ hashes.

6 Limitations and Open Problems

Improving Concrete Efficiency. We present our StE construction as a proof of concept — showing how constant round and poly-logarithmic complexity could be achived for updatable PSI with minimal leakage. We leave it as future work to instantiate our updatable PSI framework with a concretely efficient StE and to benchmark it against standard PSI protocols for practical set sizes.

In particular, one could hope to implement the $\mathcal{F}_{\mathsf{clnt}}$ functionality (which has the dominant cost in our StE construction) in a more concretely efficient manner than using garbled circuits or any other generic 2PC protocol.

Improving Security. Our updatable PSI framework is limited to semi-honest security and it can only achieve two-sided output i.e., both parties receive the desired output. Extending this structured encryption based framework to stronger malicious security and allowing for only one-sided output is left as future work. It should also be noted that our ESX construction is insecure even if we allow the adversary to adaptively pick elements to insert and delete, since this could cause overflow in the ESX tree with non-negligible probability. Hence, even extending our construction to the model with a passive adversary but adaptively chosen inputs is non-trivial.

Improving ESX construction. Our construction requires $\omega(k\cdot\log|X|)$ bandwidth and computation because the buckets have size $\omega(k)$ and the depth of the tree is $\Omega(\log|X|)$. Modern ORAMs like Path ORAM [SDS+18] and Onion ORAM [DvDF+16] have complexity on the order $O(\log|X|+k)$ as they work with constant-size buckets and a stash instead. Unfortunately, adapting these techniques to our setting appears difficult. The issue is that ESX inserts many items with the same assigned leaf (if x is added and deleted repeatedly). The repeated additions cannot coexist in the same internal node, but there *can* be one addition 'item' per internal node on the assigned path from the root to the leaf. In our setting this was not an issue, as the technique we applied focused on one bucket and showed that it did not overflow. In the setting with constant-size buckets, however, the analysis cannot avoid the dependence between the added items so easily. A more careful analysis might show that this approach is sound, but it is currently unclear how we can achieve this.

General StE and the UPSI framework. Although we design a custom StE scheme to generate our minimal leakage updatable PSI protocol, one could also consider instantiating our framework with more leaky but efficient StE constructions to derive updatable PSI protocols with varied security guarantees. These protocols could be used in applications where the security requirements

are not as stringent as minimal leakage. The main challenge would be to introduce server-side querying functionality for existing StE schemes. In Appendix C, we discuss an example instantiation of our framework with a dynamic StE based on the $\Pi^{\mathsf{dyn}}_{\mathsf{bas}}$ scheme from Cash et al [CJJ+14]. We show that we recover a simple but efficient OPRF-based updatable PSI protocol similar to existing protocols by Hazay and Lindell [HL08], with "historical" leakage similar to the protocol from Kiss et. al [KLS+17].

Updatable private set operations beyond PSI. For most PSI related privacy-preserving applications, the parties are interested in learning some function of the intersection (like cardinality and weighted sum) instead of the explicit intersection. The general version of the problem is called *private computation on set intersection* (PCSI). To the best of our knowledge, there is no known solution for updatable PCSI where both the computation and communication complexity of the protocol scale with the size of the updates instead of the entire set.

References

[ABD+21] Navid Alamati, Pedro Branco, Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Sihang Pu. Laconic private set intersection and applications. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 94–125, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany.

[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, Taormina, Italy, March 6–9, 2011. Springer, Heidelberg, Germany.

[AKM21] Ghous Amjad, Seny Kamara, and Tarik Moataz. Structured encryption secure against file injection attacks. (under submission at CRYPTO '21), 2021.

[ALOS22] Diego F. Aranha, Chuanwei Lin, Claudio Orlandi, and Mark Simkin. Laconic private set-intersection from pairings. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 111–124, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[ANSS16] Gilad Asharov, Moni Naor, Gil Segev, and Ido Shahaf. Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1101–1114, Cambridge, MA, USA, June 18–21, 2016. ACM Press.

[APRR24] Navid Alamati, Guru-Vamsi Policharla, Srinivasan Raghuraman, and Peter Rindal. Improved alternating moduli prfs and post-quantum signatures. *Cryptology ePrint Archive*, 2024.

[ASS21] Gilad Asharov, Gil Segev, and Ido Shahaf. Tight tradeoffs in searchable symmetric encryption. *Journal of Cryptology*, 34(2):9, April 2021.

[BMO17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1465–1482, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

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

[Bos16] Raphael Bost. Sophos - forward secure searchable encryption. Cryptology ePrint Archive, Report 2016/728, 2016. `https://eprint.iacr.org/2016/728`.

[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 *32nd USENIX Security Symposium (USENIX Security 23)*, pages 301–318, 2023.

[CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 79–88, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press.

[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, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[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, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[CJJ+13] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO '13*. Springer, 2013.

[CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS 2014*, San Diego, CA, USA, February 23–26, 2014. The Internet Society.

[CK10] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 577–594, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany.

[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, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[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, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[CNR21] David Cash, Ruth Ng, and Adam Rivkin. Improved structured encryption for sql databases via hybrid indexing. In *Applied Cryptography and Network Security: 19th International Conference, ACNS 2021, Kamakura, Japan, June 21–24, 2021, Proceedings, Part II*, pages 480–510. Springer, 2021.

[CT14] David Cash and Stefano Tessaro. The locality of searchable symmetric encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 351–368, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[DIL+22] Samuel Dittmer, Yuval Ishai, Steve Lu, Rafail Ostrovsky, Mohamed Elsabagh, Nikolaos Kiourtis, Brian Schulte, and Angelos Stavrou. Streaming and unbalanced psi from function secret sharing. In *International Conference on Security and Cryptography for Networks*, pages 564–587. Springer, 2022.

[DP17] I. Demertzis and C. Papamanthou. Fast searchable encryption with tunable locality. In *ACM International Conference on Management of Data (SIGMOD '17)*, SIGMOD '17, pages 1053–1067, New York, NY, USA, 2017. ACM.

[DPP18] Ioannis Demertzis, Dimitrios Papadopoulos, and Charalampos Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 371–406, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. Pir-psi: scaling private contact discovery. *Cryptology ePrint Archive*, 2018.

[Ds17] Jack Doerner and abhi shelat. Scaling ORAM for secure computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 523–535, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[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, Tenerife, Canary Islands, Spain, January 25–28, 2010. Springer, Heidelberg, Germany.

[DvDF+16] Srinivas Devadas, Marten van Dijk, Christopher W Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. Onion oram: A constant bandwidth blowup oblivious ram. In *Theory of Cryptography: 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II 13*, pages 145–174. Springer, 2016.

[EKPE18] Mohammad Etemad, Alptekin Küpçü, Charalampos Papamanthou, and David Evans. Efficient dynamic searchable encryption with forward privacy. *PoPETs*, 2018(1):5–20, January 2018.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 303–324, Cambridge, MA, USA, February 10–12, 2005. Springer, Heidelberg, Germany.

[FJK+15] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel Rosu, and Michael Steiner. Rich queries on encrypted data: Beyond exact matches. Cryptology ePrint Archive, Report 2015/927, 2015. `https://eprint.iacr.org/2015/927`.

[GGH+13] Craig Gentry, Kenny A. Goldman, Shai Halevi, Charanjit S. Jutla, Mariana Raykova, and Daniel Wichs. Optimizing ORAM and using it efficiently for secure computation.

In Emiliano De Cristofaro and Matthew K. Wright, editors, *PETS 2013*, volume 7981 of *LNCS*, pages 1–18, Bloomington, IN, USA, July 10–12, 2013. Springer, Heidelberg, Germany.

[GGM24] Gayathri Garimella, Benjamin Goff, and Peihan Miao. Computation efficient structure aware psi from incremental function secret sharing. *Cryptology ePrint Archive*, 2024.

[GMP16a] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. TWORAM: Efficient oblivious RAM in two rounds with applications to searchable encryption. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 563–592, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[GMP16b] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. Tworam: Efficient oblivious ram in two rounds with applications to searchable encryption. In *Annual International Cryptology Conference*, pages 563–592. Springer, 2016.

[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, Virtual Event, May 10–13, 2021. Springer, Heidelberg, Germany.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[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 16–20, 2021. Springer, Heidelberg, Germany.

[GRS22] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. Structure-aware private set intersection, with applications to fuzzy matching. In *Annual International Cryptology Conference*, pages 323–352. Springer, 2022.

[GRS23] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. Malicious secure, structure-aware private set intersection. In *Annual International Cryptology Conference*, pages 577–610. Springer, 2023.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS 2012*, San Diego, CA, USA, February 5–8, 2012. The Internet Society.

[HK14] Florian Hahn and Florian Kerschbaum. Searchable encryption with secure and efficient updates. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 310–320, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

[HL08] Carmit Hazay and Yehuda Lindell. Constructions of truly practical secure protocols using standardsmartcards. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 491–500, Alexandria, Virginia, USA, October 27–31, 2008. ACM Press.

[HSW23] Laura Hetz, Thomas Schneider, and Christian Weinert. Scaling mobile private contact discovery to billions of users. *Cryptology ePrint Archive*, 2023.

[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 *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 370–389. IEEE, 2020.

[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 577–594. Springer, Heidelberg, Germany, March 15–17, 2009.

[JL10] Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 418–435, Amalfi, Italy, September 13–15, 2010. Springer, Heidelberg, Germany.

[JSZ+22] Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. Shuffle-based private set union: Faster and more secure. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2947–2964, 2022.

[KKL+17] Kee Sung Kim, Minkyu Kim, Dongsoo Lee, Je Hong Park, and Woo-Hwan Kim. Forward secure dynamic searchable symmetric encryption with efficient updates. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1449–1463, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[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, Vienna, Austria, October 24–28, 2016. ACM Press.

[KL20] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.

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

[KM17] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 94–124, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[KM18] S. Kamara and T. Moataz. SQL on Structurally-Encrypted Data. In *Asiacrypt*, 2018.

[KM22] Seny Kamara and Tarik Moataz. Design and analysis of OST. Technical report, MongoDB, 2022.

[KP13] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In Ahmad-Reza Sadeghi, editor, *FC 2013*, volume 7859 of *LNCS*, pages 258–274, Okinawa, Japan, April 1–5, 2013. Springer, Heidelberg, Germany.

[KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 965–976, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[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, Santa Clara, CA, USA, August 14–16, 2019. USENIX Association.

[KRTW19] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 636–666, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany.

[Lin17] Yehuda Lindell. How to simulate it–a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pages 277–346, 2017.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

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

[MM16] I. Miers and P. Mohassel. Io-dsse: Scaling dynamic searchable encryption to millions of indexes by improving locality. Cryptology ePrint Archive, Report 2016/830, 2016. `http://eprint.iacr.org/2016/830`.

[MMBC15] Tarik Moataz, Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. Resizable tree-based oblivious RAM. In Rainer Böhme and Tatsuaki Okamoto, editors, *FC 2015*, volume 8975 of *LNCS*, pages 147–167, San Juan, Puerto Rico, January 26–30, 2015. Springer, Heidelberg, Germany.

[PR10] Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 502–519, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

[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, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[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, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

[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, Washington, DC, USA, August 12–14, 2015. USENIX Association.

[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, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[RA18] Amanda C. Davi Resende and Diego F. Aranha. Faster unbalanced private set intersection. In Sarah Meiklejohn and Kazue Sako, editors, *FC 2018*, volume 10957 of *LNCS*, pages 203–221, Nieuwpoort, Curaçao, February 26 – March 2, 2018. Springer, Heidelberg, Germany.

[RLJR22] Lawrence Roy, Stanislav Lyakhov, Yeongjin Jang, and Mike Rosulek. Practical {Privacy-Preserving} authentication for {SSH}. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3345–3362, 2022.

[RR17] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1229–1242, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

[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, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

[RT21] Mike Rosulek and Ni Trieu. Compact and malicious private set intersection for small sets. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1166–1181, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[SDS+18] Emil Stefanov, Marten van Dijk, Elaine Shi, T-H Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: an extremely simple oblivious ram protocol. *Journal of the ACM (JACM)*, 65(4):1–26, 2018.

[SDY+18] Xiangfu Song, Changyu Dong, Dandan Yuan, Qiuliang Xu, and Minghao Zhao. Forward private searchable symmetric encryption with optimized I/O efficiency. Cryptology ePrint Archive, Report 2018/497, 2018. `https://eprint.iacr.org/2018/497`.

[SWP00] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE Symposium on Research in Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

[WCS15] Xiao Wang, Hubert Chan, and Elaine Shi. Circuit oram: On tightness of the goldreich-ostrovsky lower bound. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 850–861, 2015.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

[ZCL+23] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Linear private set union from {Multi-Query} reverse private membership test. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 337–354, 2023.

A Updatable PSI Security Proof (Theorem 3.1)

Proof. Let $i = 1, 2$ and let $\mathcal{A}$ be an efficient adversary; We must construct a simulator $\mathcal{S}$ such that

$$\Pr[\mathbf{Real}^{\mathcal{A}}_{\Omega_{\mathsf{UPSI}},i}(1^k) = 1] - \Pr[\mathbf{Ideal}^{\mathcal{A}}_{\mathcal{F},\mathcal{L},\mathcal{S},i}(1^k) = 1] = \text{negl}(k).$$

We will prove this for $i = 1$; the case of $i = 2$ is symmetric. By the assumed security of Σ and Π_{PSU}, there exist efficient simulators $\mathcal{S}^{\Sigma}_1, \mathcal{S}^{\Sigma}_2$ satisfying the conditions in Definition 2.5 and $\mathcal{S}^{\Pi_{\mathsf{PSU}}}$ satisfying the condition in Definition 2.1. Since the $\mathcal{F}_{\mathsf{PSU}}$ is stateless, we omit the state arguments in that definition, and assume $\mathcal{S}^{\Pi_{\mathsf{PSU}}}$ is stateless as well.

We will use the games $G_0 - G_{10}$ in Figures 10, 11, 12, and 13.

Game G_0. The first game, G_0, which is given on the left side of Figure 10 computes the same function as $\mathbf{Real}^{\mathcal{A}}_{\Pi,1}(1^k)$ but performs some extra computation. The game consists of a main loop common to all the games (given at the top) and an implementation of the NextV_1 routine (given below). The main loop starts by initializing persistent variables on lines 1 – 2 that are used in NextV_1 (these include some not in the original game, but they are not used yet). In the NextV_1 subroutine, lines 1–10 compute "ideal" values that are not used in this game, but will be used in future games. The line 16, 17, 20, and 21 perform some extra computation and set flags $\mathsf{bad}_0, \mathsf{bad}_1$, but these are not used elsewhere in the game. The rest of the game computes the view of party 1 by performing the computation of each party and executing the appropriate protocols.

Writing G_j for the event that G_j outputs 1, we have

$$\Pr[\mathbf{Real}^{\mathcal{A}}_{\Pi,1}(1^k) = 1] = \Pr[G_0]. \quad (1)$$

Game G_1. The next game G_1 adds the shaded code on lines 16, and 20. Since G_0 and G_1 are "identical-until-bad", we have

$$\Pr[G_0] - \Pr[G_1] \leq \Pr[B_1], \quad (2)$$

where B_1 is the event that G_0 sets the variable bad_0. We claim that since Π_{PSU} is secure (for any leakage profile),

$$\Pr[B_1] = \text{negl}(k). \quad (3)$$

We use a straightforward reduction to the $\mathcal{L}^{\Pi_{\mathsf{PSU}}}$-security of Π_{PSU} for the first party. The reduction runs $\mathcal{A}$ to get its input vector $\overrightarrow{\mathsf{in}}$ and state. The reduction then simulates the computation of the game until line 16 or 19, at which point it can compute a pair of inputs for it's own game (which will be either $\mathbf{2pcReal}_{\Pi_{\mathsf{PSU}},1}$ or $\mathbf{2pcIdeal}_{\mathcal{F}_{\mathsf{PSU}},\mathcal{L}^{\Pi_{\mathsf{PSU}}}_1,\mathcal{S}^{\Pi_{\mathsf{PSU}}}_1,1}$). It then continues the simulation assume that bad_0 was not set, i.e. that $U'_1 = S'_1 \cup S'_2$ or $W' = R'_1 \cup R'_2$. It then submits its input vector to its own game and receives $\overrightarrow{\mathsf{V}}, \overrightarrow{\mathsf{out}}$. It ignores $\overrightarrow{\mathsf{V}}$ and checks if the sets in $\overrightarrow{\mathsf{out}}$ are indeed equal to the appropriate values. If one is incorrect, it outputs 1, and otherwise it outputs 0.

In the game $\mathbf{2pcReal}_{\Pi_{\mathsf{PSU}},1}$, we have that this reduction perfectly simulates G_0 until the bad_0 is set, and outputs 1 in this case (note that the simulation may be incorrect afterwards). On the other hand, in $\mathbf{2pcIdeal}_{\mathcal{F}_{\mathsf{PSU}},\mathcal{L}^{\Pi_{\mathsf{PSU}}}_1,\mathcal{S}^{\Pi_{\mathsf{PSU}}}_1,1}$, this reduction outputs 1 with probability 0. This implies the reduction has advantage exactly equal to $\Pr[B_0]$ and hence this probability is negligible. This establishes (3).

Game G_2. The next game adds the boxed code on lines 17 and 21. By a nearly identical argument as the previous transition, we have

$$\Pr[G_1] - \Pr[G_2] = \text{negl}(k). \quad (4)$$

The only difference is that we reduce to the security of Π_{PSU} for the second party instead of the first.

GAME G_3. The next game G_3 is given on the right of Figure 10; The line numbers are consistent between G_3 and G_2. There are two types of changes in this game: First, lines 16, 17, 20, and 21 have been collapsed to always compute the corresponding values, which does not change the logic of the code.

More substantially, lines 15 and 19 replace protocol runs with the computation of ideal functionalities, leakage, and simulators. We claim that, by the $\mathcal{L}_1^{\Pi_{\mathsf{PSU}}}$-security of Π_{PSU},

$$\Pr[G_2] - \Pr[G_3] = \mathrm{negl}(k). \tag{5}$$

This is proved via a straightforward reduction. One runs $\mathcal{A}$ to get its input vector $\overrightarrow{\mathsf{in}}$, the reduction creates its own input vector by simulating the game directly as in the reduction above, obtaining V_1^5 and V_1^6 from its own game. We note that the previous transitions to G_2 were necessary for the correctness of this reduction, since it must assume that U_1', U_2', W_1', W_2' were computed itself (which the reduction can do, following lines 16, 17, 20 and 21) and not by the protocols (which must be computed by the game and not itself). This gives (6).

GAME G_4. The next game G_4 on the left of Figure 11 does not included any of the boxed or highlighted code (the line numbers are also not consistent with the previous game). Compared to G_3, it reorders code within the "paragraphs" separated by blank lines without affecting the computation. It also adds conditional statements on lines 13 and 16, but these only set flags bad_4 and bad_5 and do not change the function computed by the game. Thus G_4 is exactly the same random variable as G_3 and

$$\Pr[G_4] = \Pr[G_3]. \tag{6}$$

GAME G_5. Next, G_5 adds the shaded code on line 16. Since G_4 and G_5 are "identical-until-bad",

$$\Pr[G_4] - \Pr[G_5] \leq \Pr[B_4], \tag{7}$$

where B_4 is the event that bad_4 is set to true in G_4. We claim that, by the correctness of Σ,

$$\Pr[B_4] = \mathrm{negl}(k). \tag{8}$$

To prove this, we can construct an efficient adversary $\mathcal{A}_4$ such that $\Pr[\mathbf{Cor}_{\Sigma}^{\mathcal{A}_4}(1^k) = 1] = \Pr[B_4]$. This adversary runs $\mathcal{A}(1^k)$ to obtain $\overrightarrow{\mathsf{in}}$, and then computes its output $\overrightarrow{\mathsf{op}}$ via

$\overrightarrow{\mathsf{op}} \leftarrow \varepsilon$
For $j = 1, \ldots, |\overrightarrow{\mathsf{in}}|$:
  $((X_+, X_-), (Y_+, Y_-) \leftarrow \overrightarrow{\mathsf{in}}[j]$
  $\overrightarrow{\mathsf{op}} \leftarrow \overrightarrow{\mathsf{op}} \| (Y_+, Y_-) \| (X_+)$

In other words, for each pair of inputs $((X_+, X_-), (Y_+, Y_-))$, it creates an update operation with (Y_+, Y_-) followed by a query operation with X_+. By construction, $\mathbf{Cor}_{\Sigma}^{\mathcal{A}_4}(1^k) = 1$ with probability $\Pr[B_4]$, showing this value is negligible. Once again, the simulation is only correct until the first bad event, but the claim still holds.

GAME G_6. The next game G_6 adds a similar reassignment after bad_5 is set to true on line 13, and similar to before we have

$$\Pr[G_5] - \Pr[G_6] \leq \Pr[B_5], \tag{9}$$

where B_5 is the event that bad_5 is set to true in G_5. A similar argument to the previous transition shows that

$$\Pr[B_5] = \text{negl}(k). \tag{10}$$

GAME G_7. We now consider G_7, on the right side of Figure 11 (without the shaded code). This game consolidates some code but computes the same random variable as G_7. In particular, it does the following:

- Lines 13 is removed, and later references to S'_2 are replaced references to S_2.
- Lines 16 is removed, and later references to S'_1 instead use S_1.
- Line 17 is deleted, and later usage of U'_1 and U'_2 is replaced by U. These values were always equal since $S'_1 = S_1$ and $S'_2 = S_2$.

After these changes, G_7 adds a new (inconsequential) check on its line 18 that does not include the shaded code. We have

$$\Pr[G_7] = \Pr[G_6]. \tag{11}$$

GAME G_8. The next game reassigns the values of R'_1, R'_2 on line 18. As before, we have

$$\Pr[G_7] - \Pr[G_8] \leq \Pr[B_7], \tag{12}$$

where B_7 is the event that G_7 sets bad_7. We claim that

$$\Pr[B_7] = 0. \tag{13}$$

We prove this by induction on the number of iterations of the main loop (i.e. the "for" loop on line 5 of the main procedure in upper left of Figure 10). The inductive claim is that, in just before the start of each iteration of NEXTV$_1$, we have $I'_1 = I'_2 = X \cap Y$. This implies the claim that $\Pr[B_7] = 0$, since if the inductive claim holds clearly $R'_1 = R_1$ and $R'_2 = R_2$ in the iteration.

For the base of the induction, before the first iteration we have that $I'_1 = I'_2 = \emptyset = X \cap Y$ based on the variables set in the main procedure before the loop starts.

For the inductive step, the claim holds for one iteration; we will show that it holds for the next iteration as well.

The current iteration updates X to $X_{\text{new}} = (X \cup X_+) \setminus X_-$ and Y to $Y_{\text{new}} = (Y \cup Y_+) \setminus Y_-$. By the inductive hypothesis, it also updates I'_1 and I'_2 to $I'_1 = (X \cap Y) \setminus W'_1 \cup U$ and $I'_2 = (X \cap Y) \setminus W'_2 \cup U$. Thus we must show that

$$(X \cap Y) \setminus W'_i \cup U = X_{\text{new}} \cap Y_{\text{new}}$$

for $i = 1, 2$. This follows by elementary set algebra:

$$\begin{aligned}
(X \cap Y) \setminus W'_i \cup U &= (X \cap Y) \setminus (R'_1 \cup R'_2) \cup U \\
&= (X \cap Y) \setminus ((X_- \cap (X \cap Y) \cup (Y_- \cap (X \cap Y)) \cup U \\
&= (X \cap Y) \setminus ((X_- \cup Y_-) \cap (X \cap Y)) \cup U \\
&= (X \cap Y) \setminus (X_- \cup Y_-) \cup U \\
&= (X_{\text{new}} \cap Y_{\text{new}}) \setminus (X_+ \cup Y_+) \cup U \\
&= (X_{\text{new}} \cap Y_{\text{new}}) \setminus (X_+ \cup Y_+) \cup (X_{\text{new}} \cap Y_+) \cup (Y_{\text{new}} \cap X_+) \\
&= X_{\text{new}} \cap Y_{\text{new}}.
\end{aligned}$$

The second equality uses the inductive hypothesis to replace R_1', R_2', the sixth equality uses the assumption that $X_+ \cap X_- = Y_+ \cap Y_- = \emptyset$, and the fifth equality uses the identity

$$(X \cap Y) \setminus (X_- \cup Y_-) = (X_{\text{new}} \cap Y_{\text{new}}) \setminus (X_+ \cup Y_+)$$

which can be seen via

$$\begin{aligned} X_{\text{new}} \cap Y_{\text{new}} \setminus (X_+ \cup Y_+) &= ((X \setminus X_-) \cup X_+) \cap ((Y \setminus Y_-) \cup Y_+)) \setminus (X_+ \cup Y_+) \\ &= (X \setminus X_-) \cap (Y \setminus Y_-) \\ &= (X \cap Y) \setminus (X_- \cup Y_-). \end{aligned}$$

This completes the proof of (13).

Game G_9. The next game removes lines 19 – 21 and replaces all usage of R_1', R_2' with R_1, R_2 respectively, usage of W_1', W_2' with W and usage of. In the resulting game, I_1', I_2' are no longer used, so lines 24 – 25 are also removed.

The game also changes the three shaded lines to compute S_1, U, and W in a way that will enable simulation. We claim that these always result in the same values, so

$$\Pr[G_9] = \Pr[G_8]. \tag{14}$$

We consider S_1, U and W individually in order. Previously, S_1 was set to $Y \cap X_+$, and now it is set to $I_{\text{cur}} \cap X_+ = (X \cap Y) \cap X_+$; These are equal because at this point in the code we always have $X_+ \subseteq X$ (this again relies on X_+ and X_- being disjoint).

Next, U is now computed as $I_{\text{cur}} \setminus I_{\text{prev}}$ instead of $S_1 \cup S_2$. To see that these are equal, take the values X, Y *at the start of the loop* and write $X_{\text{new}} = ((X \cup X_+) \setminus X_-)$ and $Y_{\text{new}} = ((Y \cup Y_+) \setminus Y_-)$. Then the following hold:

$$\begin{aligned} S_1 \cup S_2 &= (Y_{\text{new}} \cap X_+) \cup (X_{\text{new}} \cap Y_+) \\ I_{\text{cur}} \setminus I_{\text{prev}} &= X_{\text{new}} \cap Y_{\text{new}} \setminus (X \cap Y). \end{aligned}$$

An elementary argument shows these are equal: For one direction, suppose $a \in X_{\text{new}} \cap Y_{\text{new}} \setminus (X \cap Y)$. Then $a \in X_+ \cup Y_+$, so $a \in Y_{\text{new}} \cap X_+$ or $a \in X_{\text{new}} \cap Y_+$. For the other direction, suppose without loss of generality that $a \in Y_{\text{new}} \cap X_+$. Since $a \in X_+$, $a \notin X$ and $a \in X_{\text{new}}$, and we have $a \in (X_{\text{new}} \cap Y_{\text{new}}) \setminus (X \cap Y)$ as desired.

Finally, W is computed as $I_{\text{prev}} \setminus I_{\text{cur}}$ instead of $R_1 \cup R_2$. This are seen to be equal by yet more elementary set theory: We have

$$\begin{aligned} R_1 \cup R_2 &= (X_- \cap (X \cap Y)) \cup (Y_- \cap (X \cap Y)) \\ &= (X \cap Y) \cap (X_- \cup Y_-) \end{aligned}$$

and

$$I_{\text{prev}} \setminus I_{\text{cur}} = (X \cap Y) \setminus (((X \cup X_+) \setminus X_-) \cap ((Y \cup Y_+) \setminus Y_-)).$$

By assumption, X_+ is disjoint from X and Y_+ is disjoint from Y, so this is equal to

$$(X \cap Y) \setminus ((X \setminus X_-) \cap (Y \setminus Y_-)),$$

which is equal to $(X \cap Y) \cap (X_- \cup Y_-)$, as desired.

This establishes (14).

GAME G_{10}. This game only changes lines 11 and 12 (line numbers are consistent with the previous game). On line 11, instead of running the Upd^r protocol to generate an updated ES_Y and V_1^3, it uses the leakage function and simulator. A similar change is made on line 12. We claim that by the $\mathcal{L}^{\Sigma}$-security of Σ,

$$\Pr[G_9] - \Pr[G_{10}] = \mathrm{negl}(k). \tag{15}$$

This is proved via a straightforward reduction to the server's security guaranteed by Σ. The adversary can simulate the entire game except for lines 11 and 12. Since the output values ES_X, S_2' are not used anywhere else, there is no correctness issue with the non-adaptive adversary computing all of the input values up front.

GAME G_{11}. This game makes a transition similar to the previous one, on lines 13 and 14. Once again the reduction is straightforward (this time to the client's security). We have

$$\Pr[G_{10}] - \Pr[G_{11}] = \mathrm{negl}(k). \tag{16}$$

We complete the game hopping by observing that

$$\Pr[G_{11}] = \Pr[\mathbf{Ideal}^{\mathcal{A}}_{\mathcal{F}_{\mathsf{UPSI}},\mathcal{L},\mathcal{S},1}(1^k) = 1]. \tag{17}$$

This can be seen by pasting in the code of $\mathcal{F}_{\mathsf{UPSI}}$ along with $\mathcal{L}$ and $\mathcal{S}$ from the theorem statement into **Ideal**, which produces a game equivalent to G_{11}. The only differences are that some values are computed by more than algorithm (but in the same way), and the values are computed in a different, ultimately equivalent, order.

The theorem now follow by collecting (1) – (17) and observing that the sum of a constant number of negligible functions is negligible. □

Game $G_i(1^k)$ $G_0 - G_{10}$

1. $X, Y, I'_1, I'_2 \leftarrow \emptyset$
2. $\mathsf{ES}_X, \mathsf{ES}_Y, \mathsf{st}_X, \mathsf{st}_Y \leftarrow \perp$
3. $\mathsf{st}_{\mathcal{L}_1^\Sigma}, \mathsf{st}_{\mathcal{L}_2^\Sigma}, \mathsf{st}_{\mathcal{S}_1^\Sigma}, \mathsf{st}_{\mathcal{S}_2^\Sigma} \leftarrow \perp$
4. $(\overrightarrow{\mathsf{in}}, \mathsf{st}_\mathcal{A}) \xleftarrow{\$} \mathcal{A}(1^k)$
5. For $v = 1, \ldots, |\overrightarrow{\mathsf{in}}|$:
6. $\quad ((X_+, X_-), (Y_+, Y_-)) \leftarrow \overrightarrow{\mathsf{in}}[v]$
7. $\quad X_+ \leftarrow X_+ \setminus (X_- \cup X);\ X_- \leftarrow X_- \cap X$
8. $\quad Y_+ \leftarrow Y_+ \setminus (Y_- \cup Y);\ Y_- \leftarrow Y_- \cap Y$
9. $\quad \mathsf{V} \leftarrow \mathrm{NextV}_1((X_+, X_-), (Y_+, Y_-))$
10. $\quad \overrightarrow{\mathsf{V}} \leftarrow \overrightarrow{\mathsf{V}} \| \mathsf{V}$
11. $b \xleftarrow{\$} \mathcal{A}(\overrightarrow{\mathsf{V}}, \mathsf{st}_\mathcal{A})$
12. Return b

Oracle $\mathrm{NextV}_1((X_+, X_-), (Y_+, Y_-))$ G_0, G_1, $\boxed{G_2}$

1. $I_{\mathrm{prev}} \leftarrow X \cap Y$
2. $X \leftarrow (X \cup X_+) \setminus X_-$
3. $Y \leftarrow (Y \cup Y_+) \setminus Y_-$
4. $I_{\mathrm{cur}} \leftarrow X \cap Y$
5. $S_1 \leftarrow Y \cap X_+$
6. $S_2 \leftarrow X \cap Y_+$
7. $U \leftarrow S_1 \cup S_2$
8. $R_1 \leftarrow X_- \cap I_{\mathrm{prev}}$
9. $R_2 \leftarrow Y_- \cap I_{\mathrm{prev}}$
10. $W \leftarrow R_1 \cup R_2$
11. $(\mathsf{st}_X; \mathsf{ES}_X | \mathsf{V}_1^1, \mathsf{V}_2^1) \xleftarrow{\$} \mathsf{Upd}(\mathsf{st}_X, X_+, X_-; \mathsf{ES}_X)$
12. $(\perp; S'_2 | \mathsf{V}_1^2, \mathsf{V}_2^2) \xleftarrow{\$} \mathsf{SQry}(\mathsf{st}_X; Y_+, \mathsf{ES}_X)$
13. $(\mathsf{ES}_Y; \mathsf{st}_Y | \mathsf{V}_1^3, \mathsf{V}_2^3) \xleftarrow{\$} \mathsf{Upd}^r(\mathsf{ES}_Y; \mathsf{st}_Y, Y_+, Y_-)$
14. $(S'_1; \perp | \mathsf{V}_1^4, \mathsf{V}_2^4) \xleftarrow{\$} \mathsf{SQry}^r(X_+, \mathsf{ES}_Y; \mathsf{st}_Y)$
15. $(U'_1; U'_2 | \mathsf{V}_1^5, \mathsf{V}_2^5) \xleftarrow{\$} \Pi_{\mathsf{PSU}}(S'_1; S'_2)$
16. If $U'_1 \neq S'_1 \cup S'_2$: $\mathsf{bad}_0 \leftarrow$ `True` $U'_1 \leftarrow S'_1 \cup S'_2$
17. If $U'_2 \neq S'_1 \cup S'_2$: $\mathsf{bad}_1 \leftarrow$ `True` $\boxed{U'_2 \leftarrow S'_1 \cup S'_2}$
18. $R'_1 \leftarrow X_- \cap I'_1; R'_2 \leftarrow Y_- \cap I'_2$
19. $(W'_1; W'_2 | \mathsf{V}_1^6, \mathsf{V}_2^6) \xleftarrow{\$} \Pi_{\mathsf{PSU}}(R'_1; R'_2)$
20. If $W'_1 \neq R'_1 \cup R'_2$: $\mathsf{bad}_0 \leftarrow$ `True` $W'_1 \leftarrow R'_1 \cup R'_2$
21. If $W'_2 \neq R'_1 \cup R'_2$: $\mathsf{bad}_1 \leftarrow$ `True` $\boxed{W'_2 \leftarrow R'_1 \cup R'_2}$
22. $I'_1 \leftarrow (I'_1 \setminus W'_1) \cup U'_1$
23. $I'_2 \leftarrow (I'_2 \setminus W'_2) \cup U'_2$
24. Return $(\mathsf{V}_1^j)_{j=1}^6$

Oracle $\mathrm{NextV}_1((X_+, X_-), (Y_+, Y_-))$ G_3

1. $I_{\mathrm{prev}} \leftarrow X \cap Y$
2. $X \leftarrow (X \cup X_+) \setminus X_-$
3. $Y \leftarrow (Y \cup Y_+) \setminus Y_-$
4. $I_{\mathrm{cur}} \leftarrow X \cap Y$
5. $S_1 \leftarrow Y \cap X_+$
6. $S_2 \leftarrow X \cap Y_+$
7. $U \leftarrow S_1 \cup S_2$
8. $R_1 \leftarrow X_- \cap I_{\mathrm{prev}}$
9. $R_2 \leftarrow Y_- \cap I_{\mathrm{prev}}$
10. $W \leftarrow R_1 \cup R_2$
11. $(\mathsf{st}_X; \mathsf{ES}_X | \mathsf{V}_1^1, \mathsf{V}_2^1) \xleftarrow{\$} \mathsf{Upd}(\mathsf{st}_X, X_+, X_-; \mathsf{ES}_X)$
12. $(\perp; S'_2 | \mathsf{V}_1^2, \mathsf{V}_2^2) \xleftarrow{\$} \mathsf{SQry}(\mathsf{st}_X; Y_+, \mathsf{ES}_X)$
13. $(\mathsf{ES}_Y; \mathsf{st}_Y | \mathsf{V}_1^3, \mathsf{V}_2^3) \xleftarrow{\$} \mathsf{Upd}^r(\mathsf{ES}_Y; \mathsf{st}_Y, Y_+, Y_-)$
14. $(S'_1; \perp | \mathsf{V}_1^4, \mathsf{V}_2^4) \xleftarrow{\$} \mathsf{SQry}^r(X_+, \mathsf{ES}_Y; \mathsf{st}_Y)$
15. $U'_1 \leftarrow S'_1 \cup S'_2$
    $\lambda^5 \xleftarrow{\$} \mathcal{L}_1^{\Pi_{\mathsf{PSU}}}(S'_1, S'_2)$
    $\mathsf{V}_1^5 \xleftarrow{\$} \mathcal{S}_1^{\Pi_{\mathsf{PSU}}}(S'_1, U'_1, \lambda^5)$
16. $U'_1 \leftarrow S'_1 \cup S'_2$
17. $U'_2 \leftarrow S'_1 \cup S'_2$
18. $R'_1 \leftarrow X_- \cap I'_1; R'_2 \leftarrow Y_- \cap I'_2$
19. $W'_1 \leftarrow R'_1 \cup R'_2$;
    $\lambda^6 \xleftarrow{\$} \mathcal{L}_1^{\Pi_{\mathsf{PSU}}}(R'_1, R'_2)$
    $\mathsf{V}_1^6 \xleftarrow{\$} \mathcal{S}_1^{\Pi_{\mathsf{PSU}}}(R'_1, W'_1, \lambda^6)$
20. $W'_1 \leftarrow R'_1 \cup R'_2$
21. $W'_2 \leftarrow R'_1 \cup R'_2$
22. $I'_1 \leftarrow (I'_1 \setminus W'_1) \cup U'_1$
23. $I'_2 \leftarrow (I'_2 \setminus W'_2) \cup U'_2$
24. Return $(\mathsf{V}_1^j)_{j=1}^6$

Figure 10: The main code for all games $G_0 - G_{11}$ used in the proof of Theorem 3.1 is given in the upper left; The procedure NextV_1 changes between each game, and the implementation for games G_0, G_1,and G_2 is given on the left and G_3 is given on the right. On the left side, G_0 includes neither the shaded code nor the boxed code; G_1 adds the shaded code, and G_2 adds the boxed code to G_1. On the right, G_3 contains the shaded code, which highlights the changes from G_2.

Oracle $\textsc{NextV}_1((X_+, X_-), (Y_+, Y_-))$ G_4, G_5, $\boxed{G_6}$

1. $I_{\text{prev}} \leftarrow X \cap Y$
2. $X \leftarrow (X \cup X_+) \setminus X_-$
3. $Y \leftarrow (Y \cup Y_+) \setminus Y_-$
4. $I_{\text{cur}} \leftarrow X \cap Y$
5. $S_1 \leftarrow Y \cap X_+$
6. $S_2 \leftarrow X \cap Y_+$
7. $U \leftarrow S_1 \cup S_2$
8. $R_1 \leftarrow X_- \cap I_{\text{prev}}$
9. $R_2 \leftarrow Y_- \cap I_{\text{prev}}$
10. $W \leftarrow R_1 \cup R_2$
11. $(\mathsf{st}_X; \mathsf{ES}_X | \mathsf{V}_1^1, \mathsf{V}_2^1) \xleftarrow{\$} \mathsf{Upd}(\mathsf{st}_X, X_+, X_-; \mathsf{ES}_X)$
12. $(\perp; S_2' | \mathsf{V}_1^2, \mathsf{V}_2^2) \xleftarrow{\$} \mathsf{SQry}(\mathsf{st}_X; Y_+, \mathsf{ES}_X)$
13. If $S_2' \neq S_2$: $\mathsf{bad}_5 \leftarrow \mathtt{True}$; $\boxed{S_2' \leftarrow S_2}$
14. $(\mathsf{ES}_Y; \mathsf{st}_Y | \mathsf{V}_1^3, \mathsf{V}_2^3) \xleftarrow{\$} \mathsf{Upd}^r(\mathsf{ES}_Y; \mathsf{st}_Y, Y_+, Y_-)$
15. $(S_1'; \perp | \mathsf{V}_1^4, \mathsf{V}_2^4) \xleftarrow{\$} \mathsf{SQry}^r(X_+, \mathsf{ES}_Y; \mathsf{st}_Y)$
16. If $S_1' \neq S_1$: $\mathsf{bad}_4 \leftarrow \mathtt{True}$; $S_1' \leftarrow S_1$
17. $U_1' \leftarrow S_1' \cup S_2'; U_2' \leftarrow S_1' \cup S_2'$
18. $\lambda^5 \xleftarrow{\$} \mathcal{L}_1^{\Pi_{\mathsf{PSU}}}(S_1', S_2')$
19. $\mathsf{V}_1^5 \xleftarrow{\$} \mathcal{S}_1^{\Pi_{\mathsf{PSU}}}(S_1', U_1', \lambda^7)$
20. $R_1' \leftarrow X_- \cap I_1'; R_2' \leftarrow Y_- \cap I_2'$
21. $W_1' \leftarrow R_1' \cup R_2'; W_2' \leftarrow R_1' \cup R_2'$
22. $\lambda^6 \xleftarrow{\$} \mathcal{L}_1^{\Pi_{\mathsf{PSU}}}(R_1', R_2')$
23. $\mathsf{V}_1^6 \xleftarrow{\$} \mathcal{S}_1^{\Pi_{\mathsf{PSU}}}(R_1', W_1', \lambda^6)$
24. $I_1' \leftarrow (I_1' \setminus W_1') \cup U_1'$
25. $I_2' \leftarrow (I_2' \setminus W_2') \cup U_2'$
26. Return $(\mathsf{V}_1^j)_{j=1}^6$

Oracle $\textsc{NextV}_1((X_+, X_-), (Y_+, Y_-))$ G_7, G_8

1. $I_{\text{prev}} \leftarrow X \cap Y$
2. $X \leftarrow (X \cup X_+) \setminus X_-$
3. $Y \leftarrow (Y \cup Y_+) \setminus Y_-$
4. $I_{\text{cur}} \leftarrow X \cap Y$
5. $S_1 \leftarrow Y \cap X_+$
6. $S_2 \leftarrow X \cap Y_+$
7. $U \leftarrow S_1 \cup S_2$
8. $R_1 \leftarrow X_- \cap I_{\text{prev}}$
9. $R_2 \leftarrow Y_- \cap I_{\text{prev}}$
10. $W \leftarrow R_1 \cup R_2$
11. $(\mathsf{st}_X; \mathsf{ES}_X | \mathsf{V}_1^1, \mathsf{V}_2^1) \xleftarrow{\$} \mathsf{Upd}(\mathsf{st}_X, X_+, X_-; \mathsf{ES}_X)$
12. $(\perp; S_2' | \mathsf{V}_1^2, \mathsf{V}_2^2) \xleftarrow{\$} \mathsf{SQry}(\mathsf{st}_X; Y_+, \mathsf{ES}_X)$
13. $(\mathsf{ES}_Y; \mathsf{st}_Y | \mathsf{V}_1^3, \mathsf{V}_2^3) \xleftarrow{\$} \mathsf{Upd}^r(\mathsf{ES}_Y; \mathsf{st}_Y, Y_+, Y_-)$
14. $(S_1'; \perp | \mathsf{V}_1^4, \mathsf{V}_2^4) \xleftarrow{\$} \mathsf{SQry}^r(X_+, \mathsf{ES}_Y; \mathsf{st}_Y)$
15. $\lambda^5 \xleftarrow{\$} \mathcal{L}_1^{\Pi_{\mathsf{PSU}}}(S_1, S_2)$
16. $\mathsf{V}_1^5 \xleftarrow{\$} \mathcal{S}_1^{\Pi_{\mathsf{PSU}}}(S_1, U, \lambda^7)$
17. $R_1' \leftarrow X_- \cap I_1'; R_2' \leftarrow Y_- \cap I_2'$
18. If $R_1' \neq R_1$ or $R_2' \neq R_2$:
    $\mathsf{bad}_7 \leftarrow \mathtt{True}$; $R_1' \leftarrow R_1$; $R_2' \leftarrow R_2$
19. $W_1' \leftarrow R_1' \cup R_2'; W_2' \leftarrow R_1' \cup R_2'$
20. $\lambda^6 \xleftarrow{\$} \mathcal{L}_1^{\Pi_{\mathsf{PSU}}}(R_1', R_2')$
21. $\mathsf{V}_1^6 \xleftarrow{\$} \mathcal{S}_1^{\Pi_{\mathsf{PSU}}}(R_1', W_1', \lambda^6)$
22. $I_1' \leftarrow (I_1' \setminus W_1') \cup U$
23. $I_2' \leftarrow (I_2' \setminus W_2') \cup U$
24. Return $(\mathsf{V}_1^j)_{j=1}^6$

Figure 11: Implementation of $\textsc{NextV}_1$ for games $G_4 - G_6$ (on the left) and games G_7, G_8 (on the right), used in the proof of Theorem 3.1. Game G_4 contains neither the shaded nor the boxed code; G_4 adds the shaded code and G_5 adds the boxed code. On the right-hand side, game G_7 does not contain the shaded code; It is added to form G_8.

```
Oracle NextV_1((X_+, X_-), (Y_+, Y_-))   G_9
1  I_prev ← X ∩ Y
2  X ← (X ∪ X_+) \ X_-
3  Y ← (Y ∪ Y_+) \ Y_-
4  I_cur ← X ∩ Y
5  S_1 ← I_cur ∩ X_+
6  S_2 ← X ∩ Y_+
7  U ← I_cur \ I_prev
8  R_1 ← X_- ∩ I_prev
9  R_2 ← Y_- ∩ I_prev
10 W ← I_prev \ I_cur

11 (st_X; ES_X | V_1^1, V_2^1) <-$ Upd(st_X, X_+, X_-; ES_X)
12 (⊥; S'_2 | V_1^2, V_2^2) <-$ SQry(st_X; Y_+, ES_X)

13 (ES_Y; st_Y | V_1^3, V_2^3) <-$ Upd^r(ES_Y; st_Y, Y_+, Y_-)
14 (S'_1; ⊥ | V_1^4, V_2^4) <-$ SQry^r(X_+, ES_Y; st_Y)

15 λ^5 <-$ L_1^{Π_PSU}(S_1, S_2)
16 V_1^5 <-$ S_1^{Π_PSU}(S_1, U, λ^5)

17 λ^6 <-$ L_1^{Π_PSU}(R_1, R_2)
18 V_1^6 <-$ S_1^{Π_PSU}(R_1, W, λ^6)
19 Return (V_1^j)_{j=1}^6
```

```
Oracle NextV_1((X_+, X_-), (Y_+, Y_-))   G_10
1  I_prev ← X ∩ Y
2  X ← (X ∪ X_+) \ X_-
3  Y ← (Y ∪ Y_+) \ Y_-
4  I_cur ← X ∩ Y
5  S_1 ← I_cur ∩ X_+
6  S_2 ← X ∩ Y_+
7  U ← I_cur \ I_prev
8  R_1 ← X_- ∩ I_prev
9  R_2 ← Y_- ∩ I_prev
10 W ← I_prev \ I_cur

11 (λ^1, st_{L_1^Σ}) <-$ L_1^Σ(X_+, X_-, st_{L_1^Σ})
   (V_1^1, st_{S_1^Σ}) <-$ S_1^Σ((X_+, X_-), λ^1, st_{S_1^Σ})
12 (λ^2, st_{L_1^Σ}) <-$ L_1^Σ(Y_+, st_{L_1^Σ})
   (V_1^2, st_{S_1^Σ}) <-$ S_1^Σ(λ^2, st_{S_1^Σ})

13 (ES_Y; st_Y | V_1^3, V_2^3) <-$ Upd^r(ES_Y; st_Y, Y_+, Y_-)
14 (S'_1; ⊥ | V_1^4, V_2^4) <-$ SQry^r(X_+, ES_Y; st_Y)

15 λ^5 <-$ L_1^{Π_PSU}(S_1, S_2)
16 V_1^5 <-$ S_1^{Π_PSU}(S_1, U, λ^5)

17 λ^6 <-$ L_1^{Π_PSU}(R_1, R_2)
18 V_1^6 <-$ S_1^{Π_PSU}(R_1, W, λ^6)
19 Return (V_1^j)_{j=1}^6
```

Figure 12: Implementation of $\textsc{NextV}_1$ for games G_9 (on the left) and G_{10} (on the right). Both games included the shaded code, which highlights the difference from the preceding game.

B ESX Proof (Lemma 4.2)

Proof. For the first invariant, we observe that during an epoch, each operation adds one item to the tree, and possibly deletes some previous ones. While we cannot control the exact time at which the previous items are deleted, we know that all deletes from the previous epoch will be deleted by the end of the epoch, and each such delete will remove two items from the tree. Thus each epoch, in the worst case, increases the number of elements by the length of the current epoch divided by 8 and decreases by double the number of deletes in the previous epoch. The claim then follows.

We prove the second invariant and third invariant together by induction. They both clearly hold at the start, when the first epoch has $h_0 = h_1 = 0$ and there are no items stored in the tree. Now suppose the invariants hold at the end of some epoch with a tree of height h. During the next epoch, there are three possibilities: h is either increased, decreased, or unchanged. We consider these separately:

1. If h is decreased, then $h_0 = h, h_1 = h - 1$ during this epoch. Since this epoch will decrease h, the number of items in the tree is at most $2^h/8$ at the start of the epoch. The next epoch will have length $2^{h-1}/8 = 2^h/16$, and hence add that many items to tree.

   To establish the second invariant, we must show that the load never exceeds $2^{\min\{h_0,h_1\}} = 2^h/2$, and to establish the third, we must show that the load is at most $2^{h_1-1} = 2^h/4$ at the end of the epoch. But the load on the tree never exceeds

   $$2^h/8 + 2^h/16 < 2^h/4,$$

   which shows that both invariants hold for the case of a decreasing epoch.

Oracle $\textsc{NextV}_1((X_+, X_-), (Y_+, Y_-))$ G_{11}

1 $I_{\text{prev}} \leftarrow X \cap Y$
2 $X \leftarrow (X \cup X_+) \setminus X_-$
3 $Y \leftarrow (Y \cup Y_+) \setminus Y_-$
4 $I_{\text{cur}} \leftarrow X \cap Y$
5 $S_1 \leftarrow I_{\text{cur}} \cap X_+$
6 $S_2 \leftarrow X \cap Y_+$
7 $U \leftarrow I_{\text{cur}} \setminus I_{\text{prev}}$
8 $R_1 \leftarrow X_- \cap I_{\text{prev}}$
9 $R_2 \leftarrow Y_- \cap I_{\text{prev}}$
10 $W \leftarrow I_{\text{prev}} \setminus I_{\text{cur}}$

11 $(\lambda^1, \mathsf{st}_{\mathcal{L}_1^\Sigma}) \xleftarrow{\$} \mathcal{L}_1^\Sigma(X_+, X_-, \mathsf{st}_{\mathcal{L}_1^\Sigma})$
$(\mathsf{V}_1^1, \mathsf{st}_{\mathcal{S}_1^\Sigma}) \xleftarrow{\$} \mathcal{S}_1^\Sigma((X_+, X_-), \lambda^1, \mathsf{st}_{\mathcal{S}_1^\Sigma})$
12 $(\lambda^2, \mathsf{st}_{\mathcal{L}_1^\Sigma}) \xleftarrow{\$} \mathcal{L}_1^\Sigma(Y_+, \mathsf{st}_{\mathcal{L}_1^\Sigma})$
$(\mathsf{V}_1^2, \mathsf{st}_{\mathcal{S}_1^\Sigma}) \xleftarrow{\$} \mathcal{S}_1^\Sigma(\lambda^2, \mathsf{st}_{\mathcal{S}_1^\Sigma})$

13 $(\lambda^3, \mathsf{st}_{\mathcal{L}_2^\Sigma}) \xleftarrow{\$} \mathcal{L}_2^\Sigma(Y_+, Y_-, \mathsf{st}_{\mathcal{L}_2^\Sigma})$
$(\mathsf{V}_1^3, \mathsf{st}_{\mathcal{S}_2^\Sigma}) \xleftarrow{\$} \mathcal{S}_2^\Sigma(\lambda^3, \mathsf{st}_{\mathcal{S}_2^\Sigma})$
14 $(\lambda^4, \mathsf{st}_{\mathcal{L}_2^\Sigma}) \xleftarrow{\$} \mathcal{L}_2^\Sigma(X_+, \mathsf{st}_{\mathcal{L}_2^\Sigma})$
$(\mathsf{V}_1^4, \mathsf{st}_{\mathcal{S}_2^\Sigma}) \xleftarrow{\$} \mathcal{S}_2^\Sigma(X_+, \lambda^4, \mathsf{st}_{\mathcal{S}_2^\Sigma})$

15 $\lambda^5 \xleftarrow{\$} \mathcal{L}_1^{\Pi_{\mathsf{PSU}}}(S_1, S_2)$
16 $\mathsf{V}_1^5 \xleftarrow{\$} \mathcal{S}_1^{\Pi_{\mathsf{PSU}}}(S_1, U, \lambda^5)$

17 $\lambda^6 \xleftarrow{\$} \mathcal{L}_1^{\Pi_{\mathsf{PSU}}}(R_1, R_2)$
18 $\mathsf{V}_1^6 \xleftarrow{\$} \mathcal{S}_1^{\Pi_{\mathsf{PSU}}}(R_1, W, \lambda^6)$
19 Return $(\mathsf{V}_1^j)_{j=1}^6$

Figure 13: Implementation of $\textsc{NextV}_1$ for game G_{11}. The shaded code highlights the changes from the previous game.

2. If h stays the same, then $h_0 = h_1 = h$, and the tree holds fewer than $2^h/4$ items. The next epoch will, in the worst case, add $2^h/8$ items. We must show that the load never exceeds $2^{\min\{h_0,h_1\}} = 2^h$ and that at the end of the epoch the load is no greater than $2^h/2$. Since the epoch is not increasing the height of the tree, we know that the load is less than $2^h/8$, and during the epoch the
$$2^h/4 + 2^h/8 < 2^h/4,$$
which establishes both invariants.

3. If h increases, then $h_0 = h, h_1 = h + 1$. To establish the invariants we must show that during this epoch the load does not exceed $2^{\min\{h_0,h_1\}} = 2^h$, and that at the end of the epoch the load is less than $2^{h_1}/2 = 2^h$, i.e. prove the same bound.

   By the third invariant for the previous epoch, we know that the load on this tree is at most $2^h/2$. The next epoch will add $2^h/8$ items to the tree, in the worst case. Thus the load never exceeds
$$2^h/2 + 2^h/8 < 2^h,$$
as desired.

□

C An Example Instantiation of the Framework

As an example of how our framework can generate different updatable PSI protocols, we instantiate our general framework with a simple dynamic StE scheme with server-side querying, to recover an OPRF-based UPSI protocol.

Example: A dynamic StE scheme. Our dynamic StE scheme for encrypted sets is inspired by the $\Pi^{\mathsf{dyn}}_{\mathsf{bas}}$ scheme of Cash et al [CJJ+14]. The pseudocode for the construction is given in Figure 14. We use the following building blocks: a variable length PRF F that takes a k-bit key and produces a k-bit output, and a protocol Π_F that implements the oblivious PRF functionality for F. We also use a collision-resistant hash function H.

Outline. In brief, the encrypted set is a collection of pseudorandom tags for the elements.

To update, the client initializes its state if necessary. The routine $\mathsf{InitSt}(k)$ chooses the keys $(K_F, K_H) \xleftarrow{\$} \{0,1\}^k$. Lines 3-5 ensure that X_+, X_- are well-formed, and update the local copy of X. The client then computes hashes of all the elements of X_+ and applies the PRF to the hashes to generate a set of tags τ_+. Similarly, it generates set τ_- for X_-, and sends both τ_+ and τ_- to the server. The server simply adds all the elements of τ_+ to ES and removes all the elements of τ_-.

The server-side query protocol begins with the server initializing the encrypted set and the hash key if needed. The server hashes each element in its input set X_{qry}, and invokes the protocol Π_F with the client to get the output y. Π_F evaluates the oblivious PRF functionality, $\mathcal{F}(K_F; z)$ that outputs $(\perp; F(K_F, z))$. Next, for each pseudorandom tag y of element x, the server checks if it is present in the set ES. If it is, the server adds x to the output set.

Security. For server queries, the client only sees the number of invocations of Π_F, which corresponds to the number of queries the server makes, or the size of the set X_{qry}. For client updates, since the elements map deterministically to pseudorandom tags, the server learns the full add/delete history of any element, even if it does not know the elements themselves. Additionally, since the server queries for some elements and learns the corresponding pseudorandom tags during server-side query, the server also learns the element to tag mapping for the elements of X_{qry}.

Protocol $\mathsf{Upd}(\mathsf{st}, (X_+, X_-); \mathsf{ES})$	Protocol $\mathsf{SQry}(\mathsf{st}; X_{\text{qry}}, \mathsf{ES})$ (Server computation)
Client: 1 If $\mathsf{st} = \bot$: $\mathsf{st} \leftarrow \mathsf{InitSt}(k)$ 2 Parse st as (K_F, K_H, X) 3 $X_+ \leftarrow X_+ \setminus (X_- \cup X)$ 4 $X_- \leftarrow X_- \cap X$ 5 $X \leftarrow (X \setminus X_-) \cup X_+$ 6 $\tau_+ \leftarrow \emptyset$ 7 For $x \in X_+$: $\tau_+ \leftarrow \tau_+ \cup \{F_{K_F}(H_{K_H}(x))\}$ 8 $\tau_- \leftarrow \emptyset$ 9 For $x \in X_-$: $\tau_- \leftarrow \tau_- \cup \{F_{K_F}(H_{K_H}(x))\}$ 10 Send τ_+, τ_- Server: 11 $\mathsf{ES} \leftarrow \mathsf{ES} \cup \tau_+$ 12 $\mathsf{ES} \leftarrow \mathsf{ES} \setminus \tau_-$ 13 Output ES	1 If $\mathsf{ES} = \bot$: $\mathsf{ES} \leftarrow (\emptyset, K_H)$ 2 For $x \in X_{\text{qry}}$: 3 Initiate $(\bot; y) \xleftarrow{\$} \Pi_F(K_F; H_{K_H}(x))$ 4 If $y \in \mathsf{ES}$: $\mathsf{out} \leftarrow \mathsf{out} \cup \{x\}$ 5 Output out

Figure 14: An example construction $\Sigma = (\mathsf{Upd}, \mathsf{SQry})$.

UPSI PROTOCOL. When we instantiate our framework with our construction, we recover an updatable PSI protocol that reveals the add/delete history of every element to the party holding the encrypted set. However, the protocol is efficient in both computation and communication.