Malicious Secure, Structure-Aware Private Set Intersection

Gayathri Garimella[*] Mike Rosulek[*] Jaspal Singh[*]

Oregon State University

Abstract

Structure-Aware private set intersection (sa-PSI) is a variant of PSI where Alice's input set A has some publicly known structure, Bob's input B is an unstructured set of points, and Alice learns the intersection $A \cap B$. sa-PSI was recently introduced by Garimella *et al.* (Crypto 2022), who described a semi-honest protocol with communication that scales with the description size of Alice's set, instead of its cardinality. In this paper, we present the first sa-PSI protocol secure against malicious adversaries.

sa-PSI protocols are built from function secret sharing (FSS) schemes, and the main challenge in our work is ensuring that multiple FSS sharings encode the *same* structured set. We do so using a cut-and-choose approach. In order to make FSS compatible with cut-and-choose, we introduce a new variant of function secret sharing, called *derandomizable* FSS (dFSS).

We show how to construct dFSS for union of geometric balls, leading to a malicious-secure sa-PSI protocol where Alice's input is a union of balls. We also improve prior FSS constructions, giving asymptotic improvements to semi-honest sa-PSI.

1 Introduction

Private Set Intersection (PSI) enables two parties Alice (with private input set A) and Bob (with private input set B) to learn the intersection $A \cap B$ of their sets, while ensuring that no party learns anything beyond the intersection. All standard PSI protocols have communication and computation complexity proportional to the cardinality of the input sets [HFH99a, PSWW18, PRTY20, CM20, GPR+21, GMR+21, RS21].

Structure Aware Private Set intersection, proposed by Garimella *et al.* in [GRS22], is a variant of PSI where one party's input set has a publicly known structure. Formally, Alice has input $A \in \mathcal{S}$ from a known family $\mathcal{S}$ of sets, Bob has input B (an arbitrary set), and Alice learns $A \cap B$. The goal is for the communication cost to scale with the description size of A rather than its cardinality. The main result in [GRS22] is a generic framework that reduces structure-aware PSI to the task of designing an efficient (in terms of share size and evaluation) function secret sharing (FSS) scheme for the same family $\mathcal{S}$ of sets.

Structure-aware PSI is a powerful idea; suppose the publicly known structure is a union of n disjoint, fixed-radius balls in some distance metric. PSI of such sets solves the problem of fuzzy PSI, where Alice and Bob can identify which pairs of their points are within some small fixed distance of each other. Structure-aware PSI would have communication that scales with the description size n instead of the *total volume* of the balls.

[*]Authors partially supported by NSF award 2150726.

1.1 Our Contributions

The protocol construction in [GRS22] is secure in the semi-honest model. Our main contribution is to carefully extend their framework for structure-aware PSI to be secure in the presence of malicious adversaries. We incorporate a cut-and-choose style check to thwart any malicious behavior. For this, we require additional properties from the underlying boolean FSS and introduce a new variant called *derandomizable* function secret sharing (dFSS). We formally characterize dFSS and show new constructions.

Intuition behind Derandomizable FSS In our protocol (following the outline of [GRS22]) the receiver Alice is supposed to generate many FSS sharings of her PSI input set. A malicious Alice may generate invalid FSS shares, and/or generate FSS sharings of inconsistent sets. We want to prevent and detect this kind of misbehavior, using a cut-and-choose technique.

In a cut-and-choose, we expect Alice to generate more FSS sharings than needed, then convince Bob that most of them were generated correctly, by "opening" them — i.e., by revealing both of its shares. However, the FSS shares are shares of Alice's private input, so revealing both shares has the unpleasant side-effect of revealing Alice's private input.

We avoid this issue with the following idea: Instead of generating FSS shares of her input, Alice can generate shares of some suitable "random objects R_i." Revealing both shares reveals only R_i, and not Alice's input.

But our protocol requires FSS shares that represent Alice's input. Hence, we require an FSS scheme with the following property: Given one FSS share of R_i, and given a public "offset" value $A - R_i$, the two shareholders compute secret shares of an output that corresponds to the object A, not the object R_i. We call such an FSS scheme **derandomizable**.

Derandomizable FSS allows Bob to check that FSS shares are well-formed (by opening some of them in a cut-and-choose). However, it does not immediately help Bob check that different FSS shares encode the same object — we address that in the details of our protocol. This property of derandomizing FSS keys was first observed by Boyle et al. [BGI19] for the class of point functions, where they use this idea to reduce online complexity of secure 2 party computation of certain types of FSS gates. We formalize this derandomizable property of FSS and design it for a wider collection of sets.

Malicious-secure, structure-aware PSI We show a malicious-secure structure-aware PSI protocol in the UC model, given a dFSS for $\mathcal{S}$. Our protocol requires additively homomorphic commitments and committed oblivious transfer. The communication complexity of the protocol is $\mathcal{O}(\kappa(\sigma + d + \kappa + |B|))$, where σ is the share size of dFSS, d is the description size of a structured set, κ is the security parameter.

New dFSS definition and techniques. We introduce the notion of derandomizable bFSS (dFSS) and present the following generic transformation results. Given dFSS for collections of sets $\mathcal{S}_1, \mathcal{S}_2$ we can construct:

- dFSS for **complement** set $\overline{\mathcal{S}_1} = \{S \mid \overline{S} \in \mathcal{S}_1\}$
- dFSS for **cross product** of the sets $\mathcal{S}_1 \times \mathcal{S}_2$
- dFSS for **point function tensor product** $\mathsf{PT} \otimes \mathcal{S}_1 = \{(\mathcal{U} \setminus \{x\}) \times S | x \in \mathcal{U}, S \in \mathcal{S}\}$, where PT represents a family of singleton sets/point functions over $\mathcal{U}$.

Next, we present a new spatial hashing technique for constructing dFSS for (two classes of) disjoint union of sets:

construction	dFSS?	share size	eval cost
disjoint balls: basic FSS [BGI16]	yes	$O(n\kappa u^d)$	$O(\boldsymbol{n}u^d)$
disjoint balls: [GRS22]	no	$O(n(4\log\delta)^d\kappa)$	$O((2\log\delta)^d)$
disjoint balls: ours	yes	$O(n\kappa(ud+(\log\delta)^d))$	$O((2\log\delta)^d)$
balls with centers $>4\delta$ apart: [GRS22]	no	$O(nd2^d\kappa\log\delta)$	$O(d\log\delta)$
balls with centers $>8\delta$ apart: ours	yes	$O(nd\kappa\log\delta)$	$O(d\log\delta)$
axis-disjoint balls: [GRS22]	no	$O(nd\kappa\log\delta)$	$O(d\log\delta)$

Figure 1: FSS share size for n balls (ℓ_∞ norm) of radius δ in d dimensions, over u-bit integers. Evaluation time is for evaluating on one point.

structured input set	Communication cost (in GB)	
	[GRS22] (semi-honest secure)	Ours (malicious secure)
disjoint union of ℓ_∞ balls	101.1	662.4
union of balls with centers $>8\delta$ apart	13.2	85.6

Figure 2: PSI concrete communication cost comparison with [GRS22]. Here Alice's structured set (of size $\approx 10^7$) contains $n = 2700$ ℓ_∞ balls in 2 dimensions, each with radius $\delta = 30$ in universe of size 2^{32} along each dimension, and Bob inputs an unstructured set of size 10^6.

- Union of disjoint $\delta-$radius balls in ℓ_∞ metric space. Our share size is $\mathcal{O}(n\kappa(ud + (\log\delta)^d))$, where κ is the security parameter, $[2^u]^d$ is the input domain (*i.e.*, d-dimensional points with u-bit coordinates) and n is the number of balls.
- Union of $\delta-$radius balls with pairwise distance between the balls' centers $> 8\delta$ in ℓ_∞ metric space. Our share size is $\mathcal{O}(\kappa nud)$.

Finally, our new dFSS constructions can be used to improve the *semi-honest* sa-PSI protocol [GRS22]. If Alice's input is the union of n disjoint balls of fixed radius δ in ℓ_∞ metric space, we achieve the following improvement in communication -

- $\mathcal{O}(\kappa(n(4\log\delta)^d + |B|)) \to \mathcal{O}(\kappa(nu + n(\log\delta)^d + |B|))$, where the universe set is $[2^u]^d$.
- $\mathcal{O}(\kappa(nd(\log\delta)^d + |B|)) \to \mathcal{O}(\kappa(nud + |B|))$, when input balls have pairwise distance $> 8\delta$,

A more detailed comparison of FSS evaluation cost and share size is presented in Figure 1. Using these improved dFSS constructions for union of ℓ_∞ balls, our malicious secure structure-aware PSI protocol has only a $6-7\times$ communication overhead compared to the semi-honest secure protocol of [GRS22] (see Figure 2).

1.2 Related Work

Early solutions [Mea86, HFH99b] for 2-party PSI are based on Diffie-Hellman (and secure against semi-honest adversaries). In the last few years, we have seen many protocol paradigms emerge, based on techniques like key agreement [HFH99a, DT10, JL10], bloom filters [DCW13, RR17a], oblivious polynomial evaluation via additively homomorphic encryption [KS05, DMRY11], circuit-based [HEK12, PSWW18, PSTY19, GMR+21], vector oblivious linear function evaluation [RS21] to name a few.

However, truly practical and scalable solutions for PSI are mostly designed in the OT paradigm. Their efficiency comes from OT extension [IKNP03], which reduces the *marginal* cost of each OT instance to cheap symmetric-key operations (*e.g.*, calls to AES). These OT instances enable the comparisons necessary for PSI. Pinkas, Schneider, and Zohner [PSZ14] were the first to propose basing PSI directly on OT. The approach was later refined in a series of works [PSSZ15, KKRT16, RR17b, PRTY19, PRTY20, CM20, GPR$^+$21, RS21].

Structure-Aware PSI Recently, Garimella et al. [GRS22] introduced structure-aware PSI, where Alice has structured input and Bob has an unstructured set of points and the communication cost of the protocol scales with the description size instead of cardinality of the structured set. Note that *silent OT* [BCGI18, BCG$^+$19b, BCG$^+$19a, SGRR19, CRR21], which allows parties to generate essentially unlimited oblivious transfer instances with no communication, does **not** solve the problem of structure-aware PSI. Silent OT generates only *random* OT correlations, which must be converted to chosen-input OT instances using communication [Bea95], which is proportional to the cardinality of sets.

Malicious model To the best of our knowledge, the first specialized protocol (*i.e.*, with less than quadratic complexity) for PSI in the malicious setting is due to [FNP04]. Other approaches proposed for malicious 2-party PSI, use techniques like Diffie-Hellman key agreement, oblivious linear function evaluation and Homomorphic Encryption (*e.g.*, [DMRY09, HL08, JL10, DKT10, GN19]). More recent works [RR17a, RR17b, PRTY20, RS21, GPR$^+$21] use OT extension for malicious security.

FSS Function secret sharing was first introduced by Boyle et al. [BGI15], who proposed efficient FSS constructions for point functions, comparison functions and a few other interesting classes. These original FSS constructions were further optimized in a sequence of works for point functions, multi-point functions, comparison functions and d dimensional intervals [BGI16, BCG$^+$21, BGIK22]. [Weak] Boolean FSS bFSS was introduced by Garimella et al. [GRS22], and they present efficient weak bFSS constructions for union of disjoint ℓ_∞-balls and union of δ-radius ℓ_∞-balls with pairwise distance $> 4\delta$. The new spatial hashing based dFSS constructions we introduce exponentially improve on these bFSS constructions.

A number of works from the literature study the problem of verifiable function secret sharing - which allows the two parties holding FSS keys to verify their keys were correctly generated by the Share function [BGI16, dCP22, BBCG$^+$21]. However, all these works are limited to studying verifiable FSS for only point and multi-point functions, and for a special form of comparison/interval functions. Furthermore, to verify that these keys are correct with respect to an input domain D requires computational complexity proportional to $|D|$ - making this verification cost high for a large domain size. In our malicious secure PSI framework, we verify if the keys are correctly generated with communication complexity proportional to just the FSS share sizes, instead of their domain.

Boyle et al. [BGI19] study a notion of derandomizable FSS for the families of offset functions. This class includes point, comparison, interval, and d-dimensional interval functions. Our notion of dFSS requires a similar derandomizability property as well as an "extractability" property (defined in Section 3). Our definitions and constructions are not limited to families of offset functions. E.g., union of disjoint, fixed-radius d-dimensional balls is not an offset family.

2 Preliminaries

2.1 Secure Computation in the presence of Malicious Adversaries

We define security in the ideal / real world paradigm. The ideal world assumes a trusted third party that computes our chosen functionality f given the parties' choice of inputs. In the real protocol execution, the parties interact with each other according to some prescribed protocol (Π) to correctly compute the functionality f. When a *malicious* adversary corrupts a specific party, it learns its entire state, set of all received messages during execution and can cause the party to deviate arbitrarily (from expected protocol behavior) in its interaction with the honest party. In our setting, we assume the malicious adversary *statically* corrupts either the *sender* or the *receiver* at the start of the protocol. Informally, a protocol is considered secure if every attack in the real world can be simulated in the ideal world implying that the adversary never learns more than what he sees in an ideal execution of the protocol.

Definition 1. *A protocol Π is said to securely compute f in the presence of malicious adversaries if for every probabilistic polynomial time (PPT) adversary $\mathcal{A}$, there is a probabilistic polynomial time simulator Sim such that*

$$\Pr_{real}[\Pi, \mathcal{Z}, \mathcal{A}]_{x,y} \approx \Pr_{ideal}[\mathcal{F}, \mathcal{Z}, \mathsf{Sim}]_{x,y}$$

i.e. no probabilistic polynomial time distinguisher $\mathcal{Z}$ has non-negligible advantage.

2.2 Committed Oblivious Transfer

The functionality is presented in Figure 3. A sender inputs a pair of messages m_0, m_1, a receiver inputs a bit b, and the $\mathsf{transfer}$ function outputs m_b to the receiver. The open function can be invoked by the sender to reveal both messages m_0, m_1 to the receiver.

Parameter: message length l

Choose:

1. Receive (choose, id, b) from the receiver, where $b \in \{0, 1\}$.
2. If no message of the form $(\mathsf{choose}, id, .)$ is present in memory, store (choose, id, b) and send (choose, id) to the sender.

Transfer:

1. Receive $(\mathsf{transfer}, id, m_0, m_1)$ from the sender, where $m_0, m_1 \in \{0, 1\}^l$.
2. If no messages of the form $(\mathsf{transfer}, id, m_0, m_1)$ is present in memory and a message of the form (choose, id, b) is stored, send $(\mathsf{transfer}, id, m_b)$ to the receiver.

Open:

1. Receive (open, id) from the sender
2. Output stored message $(\mathsf{transfer}, id, m_0, m_1)$ to the receiver

Figure 3: Ideal functionality $\mathcal{F}_{\mathsf{COT}}$

2.3 Homomorphic Commitment Scheme

The functionality is presented in Figure 4. The functionality allows a sender and receiver to commit and open to messages. The functionality is additively homomorphic so it can reveal / open the difference of any two committed values within a session.

2.4 Cuckoo Hashing

A cuckoo hash table $\mathcal{C}$ is a randomized data structure of size m that is used to encode a set of elements (key-value pairs) X of size n parametrized by three hash functions $\mathcal{H} = (\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3)$ with

Parameter: Message space represented by some field $\mathbb{F}$.
$\mathcal{F}_{\mathsf{hcom}}$ interacts with a sender P_s, a receiver P_r and an adversary $\mathcal{S}$ and it has the following three phases:

Commit:

1. Upon receiving a message $(\mathsf{com}, sid, idx, P_s, P_r, m)$ from P_s, where $m \in \mathbb{F}$, record the tuple (idx, P_s, P_r, m) and send the message $(\mathsf{receipt}, sid, idx, P_s, P_r)$ to P_r and S.
2. Ignore any future commit messages with the same idx from P_s to P_r.
3. If a message $(\mathsf{aborts}, sid, idx)$ is received from $\mathcal{S}$, the functionality halts.

Open:

1. On receiving a message $(\mathsf{reveal}, sid, idx)$ from P_s: If a tuple (idx, P_s, P_r, m) was previously recorded, then send message $(\mathsf{ok}, sid, idx, P_s, P_r, m)$ to P_r and $\mathcal{S}$. Otherwise, ignore.
2. On receiving a message $(\mathsf{reveal}, sid, idx_0, idx_1)$ from P_s: If tuples (idx_0, P_s, P_r, m_0), (idx_1, P_s, P_r, m_1) were previously recorded, then send message $(\mathsf{ok}, sid, idx_0, idx_1, P_s, P_r, (m_0 - m_1))$ to P_r and $\mathcal{S}$. Otherwise, ignore.

Figure 4: Ideal functionality $\mathcal{F}_{\mathsf{HCom}}$

$\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3 : \{0,1\}^* \to [m]$:

$$\mathcal{C} \leftarrow \mathsf{Cuckoo}^m_{\mathsf{H}_1,\mathsf{H}_2,\mathsf{H}_3}(X)$$

A cuckoo table $\mathcal{C} = (C_1, \ldots, C_m)$ holds every element of X where for each $(x, y) \in X$ there is some $i \in \{1, 2, 3\}$ such that $C_{h_i(x)} = y$. Some positions of $\mathcal{C}$ will not matter, corresponding to empty bins. We pick the parameter m such that every element from X finds a place in the cuckoo table except with negligible probability. Concretely, we set $m = 1.27n$.

2.5 Function Secret Sharing

Boolean Function Secret Sharing [GRS22]. Weak boolean function secret sharing (p, ρ)-bFSS is a relaxation of the standard FSS definition, it can allow for FSS evaluation with output length ρ and a bounded false positive rate p. In this work, we don't allow for false positives and only consider ρ-bFSS with output length relaxation. Our full definition is in Appendix A.

3 Derandomizable bFSS

In this work, we introduce *derandomizable FSS*, a new variant of weak boolean FSS with some additional properties. First, we always translate our input into a group $\mathbb{G}$ via an encoding function. A given input can have multiple encodings, however, all of them must decode back to the same input.

Definition 2 (Encoding Function/Scheme)**.** *For a group $\mathbb{G}$, a family of sets $\mathcal{S}$, we can deterministically encode a set $\mathsf{Encode} : \mathcal{S} \to \mathbb{G}$, if there exists $\mathsf{Encode}^{-1} : \mathbb{G} \to \mathcal{S}$ such that:*

$$Pr\left(\mathsf{Encode}^{-1}(\mathsf{Encode}(S)) = S\right) = 1$$

In our ρ-dFSS definition, RShare generates key shares for a uniformly random element $R \in \mathbb{G}$ from the encoding space. The evaluation function DEval takes an additional parameter $\mathsf{offset} \in \mathbb{G}$ and evaluates the input on key shares corresponding to group element $(\mathsf{offset} + R)$ where $+$ is the group operation. Finally, we require that we can extract the group element (which encodes the structured input) from the key shares. These properties will jointly enable randomizing and de-randomizing our structured input required for our malicious-secure sa-PSI protocol.

Definition 3 (Derandomizable bFSS syntax)**.** *Let $\mathcal{S} \subseteq 2^{\mathcal{U}}$ denote a family of sets over input domain $\mathcal{U} = \mathbb{G}$, an encode function $\mathsf{Encode} : \mathcal{S} \to \mathbb{G}$ and security parameter κ. A 2-party derandomizable ρ-dFSS scheme with algorithms (RShare, Extract, DEval) has the following syntax:*

- $(k_0, k_1, R) \leftarrow \mathsf{RShare}(1^\kappa)$*: is a randomized algorithm that outputs a uniformly random group element $R \in \mathbb{G}$ and its associated (k_0, k_1) key shares.*
- $R \leftarrow \mathsf{Extract}(k_0, k_1)$*: is a deterministic function that takes key shares as input. It outputs the group element R associated with the k_0, k_1 (output by RShare).*
- $y_{\mathsf{idx}} \leftarrow \mathsf{DEval}(1^\kappa, \mathsf{idx}, k_{\mathsf{idx}}, x, \mathsf{offset})$*: is a deterministic algorithm with inputs security parameter, party index $\mathsf{idx} \in \{0,1\}$, the key share k_{idx}, the input point of evaluation $x \in \mathcal{U}$ and a group element offset $\mathsf{offset} \in \mathbb{G}$. It outputs a string $y_{\mathsf{idx}} \in \{0,1\}^\rho$.*

Definition 4 (Derandomizable bFSS security)**.** *A derandomizable 2-party ρ-dFSS scheme (RShare, Extract, DEval) for $\mathcal{S} \subseteq 2^{\mathcal{U}}$ is **secure** if it satisfies the following conditions:*

- ***Correctness for yes-instances:** For any $S \in \mathcal{S}$, $x \in S$:*

$$\Pr\left(y_0 \oplus y_1 = 0^\rho \;\middle|\; \begin{array}{l} (k_0, k_1, R) \leftarrow \mathsf{RShare}(1^\kappa) \\ y_0 \leftarrow \mathsf{DEval}(1^\kappa, k_0, x, \mathsf{Encode}(S) - R) \\ y_1 \leftarrow \mathsf{DEval}(1^\kappa, k_1, x, \mathsf{Encode}(S) - R) \end{array} \right) = 1 - \mathsf{negl.}(\kappa)$$

- ***Correctness for no-instances:** For every set $S \in \mathcal{S}$, $\underline{x \in \mathcal{U} \setminus S}$:*

$$\Pr\left(y_0 \oplus y_1 \neq 0^\rho \;\middle|\; \begin{array}{l} (k_0, k_1, R) \leftarrow \mathsf{RShare}(1^\kappa) \\ y_0 \leftarrow \mathsf{DEval}(1^\kappa, 0, k_0, x, \mathsf{Encode}(S) - R) \\ y_1 \leftarrow \mathsf{DEval}(1^\kappa, 1, k_1, x, \mathsf{Encode}(S) - R) \end{array} \right) = 1$$

- ***Privacy:** For every set $S \in \mathcal{S}$ and index $\mathsf{idx} \in \{0,1\}$ there exists a simulator Sim such that the following distributions are computationally indistinguishable in the security parameter:*

$$\boxed{\begin{array}{l} (k_0, k_1, R) \leftarrow \mathsf{RShare}(1^\kappa) \\ \text{return } (k_{\mathsf{idx}}, \mathsf{Encode}(S) - R) \end{array}} \cong_\kappa \mathsf{Sim}(1^\kappa, \mathsf{idx})$$

- ***Extractability:** there exists an efficient function Extract, such that for any pair of strings k_0, k_1 and $R \in \mathbb{G}$*

$$\perp \neq R \leftarrow \mathsf{Extract}(k_0, k_1) \iff \exists r \text{ such that } (k_0, k_1, R) \overset{r}{\leftarrow} \mathsf{RShare}(1^\kappa)$$

($\Longleftarrow$) For every output (k_0, k_1, R) of RShare, Extract outputs R given the k_0, k_1 key shares.

($\Longrightarrow$) For every output R of Extract, there exists randomness r such that RShare will output (k_0, k_1, R).

Definition 5. *A derandomizable 2-party ρ-dFSS scheme is called **a strong dFSS** if $\rho = 1$, else its referred to as a **weak dFSS**.*

Definition 6. *A derandomizable 2-party ρ-dFSS scheme for a collection of sets is said to have pseudo-random keys if the output of the simulator in Definition 4 is pseudo-random.*

4 Malicious-Secure Structure-Aware PSI

In this section, we discuss the main ideas of our malicious-secure $\mathsf{sa\text{-}PSI}$ protocol. We provide a full protocol description, proof of security and an asymptotic analysis of our communication cost.

4.1 Overview and Intuition

Let's start by reviewing the semi-honest $\mathsf{sa\text{-}PSI}$ protocol [GRS22]. The receiver (Alice) generates κ independent FSS sharings $(k_0^{(i)}, k_1^{(i)}) \leftarrow \mathsf{Share}(A)$ of her structured input A and the sender (Bob) samples a uniformly random string $s \leftarrow \{0,1\}^\kappa$. Now, both parties run κ instances of OT so that Bob learns one of the two FSS shares $k_*^{(i)} = k_{s_i}^{(i)}$ for each of his choice bits s_i. Bob can define an OPRF function as:

$$F(x) = \mathsf{H}\Big(\mathsf{Eval}(k_*^{(1)}, x), \mathsf{Eval}(k_*^{(2)}, x), \ldots, \mathsf{Eval}(k_*^{(\kappa)}, x)\Big),$$

where H is a random oracle. Bob can compute $F(x)$ for any value x of his choice. On the other hand, F is defined using Bob's choice bits s, in such a way that Alice can compute $F(x)$ only when $x \in A$. When $x \in A$, Bob's choice bits have no effect — i.e., $\mathsf{Eval}(k_0^{(i)}, x) = \mathsf{Eval}(k_1^{(i)}, x)$ — by the property of the ρ-bFSS. In this case, Alice can compute $F(x)$ as $\mathsf{H}\Big(\mathsf{Eval}(k_0^{(1)}, x), \ldots, \mathsf{Eval}(k_0^{(\kappa)}, x)\Big)$. If $x \notin A$, then each of Bob's choice bits changes one of the argument terms to H, so Alice would need to correctly guess all κ of Bob's choice bits in order to call H on the correct input to compute $F(x)$. Now we have a PSI protocol in the usual way [FIPR05]: Bob sends $\{F(b) \mid b \in B\}$ to Alice, who can only recognize those OPRF outputs that correspond to her input set A.

What goes wrong with a malicious adversary? A corrupt Alice can send incorrect or inconsistent ρ-bFSS key shares to the OT instances. Bob would like to verify that Alice's OT inputs are well-formed FSS shares consistent with her chosen structured input. A potential approach is cut-and-choose, where Bob challenges Alice with a set of randomly chosen indices to "open" (learn both key shares) and "check" (if it correctly encodes Alice's input), and aborts if any of the opened key shares are inconsistent. However, observe that a direct application of cut-and-choose violates Alice's input privacy. Bob can re-construct the input from any pair of well-formed ρ-dFSS keys he learns during the "open" phase. To get around this issue, we introduce a new variant of weak boolean FSS called *derandomizable* FSS (Section 3) with some additional useful properties.

In a dFSS scheme, the key sharing phase is randomized to generate shares of a uniformly random element (say R) from the same family of sets as the input (A). The evaluation step is modified to include an "offset" or "correction" ($A - R$) to de-randomize the evaluation from an FSS share of R to an FSS share of the chosen input (A). Within our PSI context, Alice (with chosen input $A \in \mathcal{S}$) will now generate ℓ ($> \kappa$ for cut-and-choose) dFSS shares of uniformly random elements $R_i \in \mathcal{S}$ from the same family of sets as her input. Bob picks some subset of indices ($\mathsf{OpenSet}$) and checks if the shares are well-formed and correspond to an element from $\mathcal{S}$, otherwise aborts. All the unopened OTs (termed $\mathsf{EvalSet}$ in the protocol) will be used to define the OPRF function, similar to the semi-honest protocol.

Alice sends "offset" ($A - R_i$) for all indices (in $\mathsf{EvalSet}$), so that Bob can de-randomize his received FSS shares to Alice's input. However, Alice can still cheat by de-randomizing different R_i's to different A values. To fix this, Alice must initially commit to her input encoding and all the randomly sampled R_i sets, under an additively homomorphic commitment scheme. Later, during the evaluation stage, she homomorphically decommits to the required "offsets". Now, Bob can be convinced that *all* offsets de-randomize the FSS shares to the same underlying input.

4.2 Protocol

Now, we take a closer look at the protocol and highlight some useful aspects.

Objects and Encodings We have written the "offset" value $A - R_i$, which implies that there is a group structure over the set of objects. For this reason, we make an important distinction between the primary objects (**sets** of points) and **encodings** of those objects. For example, if the set of points is a geometric ball, then a good encoding may be the coordinates of the ball's center. The group operation on encodings is vector addition of points in the ambient geometry.

The encoding of a single ball is naturally deterministic, but in other settings an object may have many valid encodings. Our dFSS construction for union of balls is such an example. Given n balls, we assign them to one of m bins using cuckoo hashing. Even with the hashing functions fixed, there are typically several legal ways to assign objects into bins using cuckoo hashing. The encoding of the set is (essentially) a vector of length m, where the ith component holds the center of the ball assigned to the ith bin (or a dummy center).

Most of the reasoning about our FSS constructions is in the "encoding domain." The PSI protocol can force a corrupt Alice to use some element of the encoding domain as her PSI input. However, not all such elements are actually encodings of a valid set. Consider the example where the valid sets are disjoint unions of n balls. Our FSS construction maps those n balls to m bins, and we must have $m = (1 + O(1)) \cdot n > n$ for cuckoo hashing to succeed with high probability for the honest parties. However, there are values in the encoding space that correspond to the union of up to m balls. Therefore, the choice of encoding determines if an adversary is able to have more balls in its input set than the honest parties.

Cut-and-Choose In Figure 6 we present our cut and choose parameters for ℓ OT instances. We allow Bob to check ℓ_1 OT instances ($\mathsf{OpenSet}$) and evaluate on the remaining ℓ_2 OT instances ($\mathsf{EvalSet}$) where $\ell = \ell_1 + \ell_2$. A corrupt Alice wins the cut-and-choose if *all* ℓ_1 OTs are correct and less than κ OT instances of $\mathsf{EvalSet}$ are correct (argument to H to compute the $\mathsf{OPRF}()$ has low entropy). In more detail, we maximize the number of indices c that are faulty so that less than κ OT instances of $\mathsf{EvalSet}$ are correct, and count the number of ways to pick $\mathsf{OpenSet}$ such that has no bad indices c out of the total number of ways to pick $\mathsf{OpenSet}$. Both these requirements must be met with negligible probability in the security parameter.

$$\max_{c : c > \ell_2 - \kappa} \frac{\binom{\ell - c}{\ell 1}}{\binom{\ell}{\ell_1}} \leq 1/2^{\lambda}$$

OPRF notation For ease of presentation, we slightly modify our notation for OPRF outputs $F(x)$. Because of cut-and-choose, only a subset of FSS instances are used for evaluation. We write the argument to the hash function H as a set rather than a string:

$$F(x) = \mathsf{H}\Big(x;\ \big\{(i, \mathsf{DEval}(k_*^{(i)}, b, \mathsf{offset}_i)) \mid i \in \mathsf{EvalSet}\big\}\Big)$$

When we write an expression like this, we mean that the items of the set (pairs) are serialized into a string in a canonical way — *i.e.*, by sorting the pairs by their first component.

Now, we present the full details of our framework for malicious-secure Structure-Aware PSI in Figure 7 and prove that our protocol securely realizes our ideal functionality in Figure 5. The protocol is in the random oracle model and makes calls to the ideal functionality for Committed OT ($\mathcal{F}_{\mathsf{COT}}$) and an Additively Homomorphic Commitment scheme ($\mathcal{F}_{\mathsf{HCom}}$).

Parameters: Given a family of subsets $\mathcal{S} \subseteq 2^{\mathcal{U}}$ over the universe $\mathcal{U}$ and an encode function $\mathsf{Encode} : \mathcal{S} \rightarrow \mathbb{G}$.

Functionality:

1. If Alice is honest, then receive from her an input set $A \in \mathcal{S}$. Otherwise if Alice is corrupt, receive from her an encoding $E \in \mathbb{G}$ and define $A = \mathsf{Encode}^{-1}(E)$.
2. Receive input $B \subseteq \mathcal{U}$ from Bob.
3. Give output $A \cap B$ and $|B|$ to Alice.

Figure 5: Ideal functionality $\mathcal{F}_{\mathsf{saPSI}}$ for Structure-Aware PSI in malicious model.

Parameters:

- Computational security parameter κ and statistical security parameter λ
- Family of sets $\mathcal{S}$ with corresponding ρ-dFSS scheme $(\mathsf{RShare}, \mathsf{Encode}, \mathsf{DEval})$
- Random oracle $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^{2\kappa}$
- Committed oblivious transfer functionality $\mathcal{F}_{\mathsf{COT}}$ (Figure 3)
- Additively homomorphic commitment functionality $\mathcal{F}_{\mathsf{HCom}}$ (Figure 4)
- $\ell = (\ell_1 + \ell_2)$ OT instances where $\max_{c:c>\ell_2-\kappa} \frac{\binom{\ell-c}{\ell_1}}{\binom{\ell}{\ell_1}} \leq 1/2^{\lambda}$. As a concrete example, $(\ell_1, \ell_2) = (60, 220)$ suffices for $(\kappa, \lambda) = (128, 40)$.

Figure 6: Parameters for Malicious-Secure Structure Aware PSI

4.3 Security against Malicious behavior

Theorem 7. *Given a ρ-dFSS scheme for a family of subsets $\mathcal{S} \subseteq 2^{\mathcal{U}}$ over input domain $\mathcal{U}$. Protocol π_{sa-PSI} (in Figure 7) UC-securely realizes $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$ in the presence of a malicious adversary in the $\mathcal{F}_{\mathsf{COT,\ HCom}}$−hybrid random oracle model.*

Proof. **Bob is corrupt**. In the ideal world execution, the simulator does the following steps to simulate a corrupt Bob's protocol view -

- Simulator observes Bob's queries to $\mathcal{F}_{\mathsf{Com}}$ in step 1 and extracts $\mathsf{OpenSet}$.
- Send acknowledgements $(\mathsf{receipt}, sid_a, \ell, P_a, P_b)$ and $\{(\mathsf{receipt}, sid_a, i, P_a, P_b) \mid i \in [\ell]\}$ to simulate the ideal functionality $\mathcal{F}_{\mathsf{HCom}}$ responses in step 2.
- Observes Bob's choice bit string s to the $\mathcal{F}_{\mathsf{COT}}$ ideal functionality. For the i^{th} OT instance -
  - If $i \in \mathsf{OpenSet}$: sample $(k_0^{(i)}, k_1^{(i)}, R_i) \leftarrow \mathsf{RShare}(1^\kappa)$.
  - If $i \notin \mathsf{OpenSet}$: call the simulator $(k_*^{(i)} = k_{s_i}^{(i)}, \mathsf{offset}_i) \leftarrow \mathsf{Sim}(1^\kappa, s_i)$ that guarantees privacy of the underlying ρ-dFSS scheme.
  - Send $(\mathsf{transfer}, i, k_*^{(i)} = k_{s_i}^{(i)})$ to Bob on behalf of $\mathcal{F}_{\mathsf{COT}}$.
  - Include $(\mathsf{ok}, sid_a, \ell, i, P_a, P_b, \mathsf{offset}_i)$ in the set of messages to Bob in step 7 of the protocol.
- For $i \in \mathsf{OpenSet}$: send $(\mathsf{transfer}, i, k_0^{(i)}, k_1^{(i)})$ and acknowledge $(\mathsf{ok}, sid_a, i, P_a, P_b, R_i)$ to Bob.
- Let $\mathsf{OPRF}(b) = \mathsf{H}\big(b;\ \{(i, \mathsf{DEval}(k_*^{(i)}, b, \mathsf{offset}_i)) \mid i \in \mathsf{EvalSet}\}\big)$. The simulator observes Bob's queries to the random oracle and defines set

$$\mathcal{B} = \{b \mid \text{Bob made a random oracle query of the form } H(b, \cdot) \text{ and } \mathsf{OPRF}(b) \in \tilde{B}\}$$

  where $\tilde{B}$ is the message sent by Bob in step 8.
- Simulator sends Bob's effective input $\mathcal{B}$ to $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$. The functionality returns $A \cap \mathcal{B}$ to Alice.

Inputs: Alice (P_a) has input $A \in \mathcal{S}$ and Bob (P_b) has input B.

Protocol:

1. Bob does the following -
   - Picks a string $s \leftarrow \{0,1\}^\ell$ uniformly at random.
   - Picks indices $\mathsf{OpenSet} \subset [\ell]$ and sends $(\mathsf{com}, sid_b, 0, P_b, P_a, \mathsf{OpenSet})$ to $\mathcal{F}_{\mathsf{Com}}$.
2. Alice makes ℓ calls to $\mathsf{RShare}(1^\kappa)$ from the ρ-dFSS scheme. Then commits to the generated sets and an encoding of her input.
   - For $i \in [\ell]$: do $(k_0^{(i)}, k_1^{(i)}, R_i) \leftarrow \mathsf{RShare}(1^\kappa)$.
   - For $i \in [\ell]$: send $(\mathsf{com}, sid_a, i, P_a, P_b, R_i)$ to $\mathcal{F}_{\mathsf{HCom}}$.
   - Alice sends $(\mathsf{com}, sid_a, \ell, P_a, P_b, \mathsf{Encode}(A))$ to $\mathcal{F}_{\mathsf{HCom}}$.
3. Alice as OT sender and Bob as OT receiver run ℓ (parallel) instances of committed oblivious transfer $\mathcal{F}_{\mathsf{COT}}$ (Figure 3). In the i^{th} instance -
   - Alice inputs the keys shares. She sends $(\mathsf{transfer}, i, k_0^{(i)}, k_1^{(i)})$ to $\mathcal{F}_{\mathsf{COT}}$.
   - Bob sends $(\mathsf{choose}, i, s_i)$ to $\mathcal{F}_{\mathsf{COT}}$ which returns $(\mathsf{transfer}, i, k_*^{(i)} = k_{s_i}^{(i)})$.
4. Bob sends $(\mathsf{reveal}, sid_b, 0)$ to $\mathcal{F}_{\mathsf{HCom}}$ to reveal $\mathsf{OpenSet}$ to Alice.
5. If Alice receives $(\mathsf{ok}, sid_b, 0, P_b, P_a, \mathsf{OpenSet})$ from $\mathcal{F}_{\mathsf{HCom}}$, then -
   - For $i \in \mathsf{OpenSet}$: send (open, i) to $\mathcal{F}_{\mathsf{COT}}$; Bob learns $(k_0^{(i)}, k_1^{(i)})$.
   - For $i \in \mathsf{OpenSet}$: send $(\mathsf{reveal}, sid_a, i)$ to $\mathcal{F}_{\mathsf{Com}}$; Bob can now learn R_i.
6. Now, Bob does the following check -
   - For $i \in \mathsf{OpenSet}$: $R_i \overset{?}{=} \mathsf{Extract}(k_0^{(i)}, k_1^{(i)})$ and aborts if function call fails.
7. Alice reveals the offsets to Bob -
   - For all $j \in \mathsf{EvalSet}$: send $(\mathsf{reveal}, sid_a, \ell, j)$ to $\mathcal{F}_{\mathsf{HCom}}$.
   - Bob receives $(\mathsf{ok}, sid_a, \ell, j, P_a, P_b, \mathsf{offset}_j)$, $\mathsf{offset}_j = (\mathsf{Encode}(A) - R_j)$.
8. Bob computes $\widetilde{B}$, defined below, and sends it (permuted uniformly at random) to Alice. Set D is serialized to a string by sorting its elements by their first components.
$$\widetilde{B} = \left\{ \mathsf{H}\big(b;\ D\big) \;\middle|\; b \in B \right\};\ \text{where } D = \{(i, \mathsf{DEval}(k_*^{(i)}, b, \mathsf{offset}_i)) \mid i \in \mathsf{EvalSet}\}$$
9. Alice computes the **PSI output** as
$$\left\{ a \in A \;\middle|\; \mathsf{H}\big(a;\ \{(i, \mathsf{DEval}(k_0^{(i)}, a, \mathsf{offset}_i)) \mid i \in \mathsf{EvalSet}\}\big) \in \widetilde{B} \right\}$$

Figure 7: Protocol description for Structure-Aware PSI in the Malicious model.

Now, we show a sequence of hybrids from the real-world to ideal world execution.

- *Hybrid 0:* We start with the real interaction where Alice interacts with her actual input A.
- *Hybrid 1:* Extract $\mathsf{OpenSet}$ from Bob's queries to $\mathcal{F}_{\mathsf{COT}}$. In this hybrid, look at indices $i \notin \mathsf{OpenSet}$: replace $(k_0^{(i)}, k_1^{(i)}, R_i) \leftarrow \mathsf{RShare}(1^\kappa)$ with $(k_{s_i}^{(i)}, \mathsf{offset}_i) \leftarrow \mathsf{Sim}(1^\kappa, s_i)$ a call to

simulator of the underlying dFSS scheme. Here, Bob's input s_i to $\mathcal{F}_{\mathsf{COT}}$ is used to simulate $k^{(i)}_{s_i}$ ($\mathcal{F}_{\mathsf{COT}}$ output sent to Bob) and offset_i (sent to Bob in step 7). The simulator knows the cut-and-choose indices $\mathsf{OpenSet}$ in advance, and hence it will never need to show the "other FSS share" for these indices. By the privacy property of the dFSS scheme (definition 3), this hybrid is indistinguishable from the real interaction.

- *Hybrid 2:* In this hybrid, we replace the actual interaction with $\mathcal{F}_{\mathsf{HCom}}$ in step 2 by sending acknowledgements $(\mathsf{receipt}, sid_a, \ell, P_a, P_b)$ and $(\mathsf{receipt}, sid_a, i, P_a, P_b)$ for all $i \in [\ell]$ to Bob, so that the actual values being committed are not used in this step. This hybrid is identically distributed with the previous hybrid since Alice is honest.
- *Hybrid 3:* Now, we modify how honest Alice computes the output of the protocol. Let $\mathsf{OPRF}(b) = \mathsf{H}\big(b;\ \{(i, \mathsf{DEval}(k^{(i)}_*, b, \mathsf{offset}_i)) \mid i \in \mathsf{EvalSet}\}\big)$. We track the queries made by Bob to the random oracle $\mathsf{OPRF}(b)$ until Bob sends $\tilde{B}$ in step 8 of the protocol. Now we define set -

$$\mathcal{B} = \{b \mid \text{Bob made a random oracle query of form } H(b, \cdot) \text{ and } \mathsf{OPRF}(b) \in \tilde{B}\}$$

and change Alice's output to $A \cap \mathcal{B}$.
It remains to show that for any $a \in A$ it belongs to modified output $A \cap \mathcal{B}$ if and only if it belongs to $A \cap B$, except with negligible probability. We have the following cases -

1. $a \in \mathcal{B}$: $\mathsf{H}\big(a;\ \{(i, \mathsf{DEval}(k^{(i)}_*, a, \mathsf{offset}_i)) \mid i \in \mathsf{EvalSet}\}\big) =$
$\mathsf{H}\big(a;\ \{(i, \mathsf{DEval}(k^{(i)}_0, a, \mathsf{offset}_i)) \mid i \in \mathsf{EvalSet}\}\big)$ if and only if $a \in A \cap B$ by the correctness property of the dFSS scheme.
2. $a \notin \mathcal{B}$ because $\mathsf{OPRF}(a) \notin \tilde{B}$: as a result $a \notin A \cap \mathcal{B}$ and $a \notin A \cap B$.
3. $a \notin \mathcal{B}$ because Bob did not query the random oracle on $\mathsf{OPRF}(a)$: here $a \notin A \cap \mathcal{B}$; but the probability that $a \in A \cap B$ is $\frac{|\tilde{B}|}{2^\kappa}$ because Alice's query to random oracle $\mathsf{OPRF}(a)$ is "freshly" random.

This hybrid differs from the previous hybrid only if case 3 is true which happens with negligible probability.

Alice is corrupt. In order to simulate Alice's view in the ideal execution, the simulator does the following:

- Sample $\mathsf{OpenSet} \subset [\ell]$ uniformly at random and send $(\mathsf{receipt}, sid_b, 0, P_b, P_a)$ to Alice. All the remaining indices are in $\mathsf{EvalSet} = [\ell] \setminus \mathsf{OpenSet}$.
- Extract $\{R_0, R_1, \ldots, R_{\ell-1}\}$ from messages $\{(\mathsf{com}, sid_a, i, P_a, P_b, R_i) \mid i \in [\ell]\}$ to $\mathcal{F}_{\mathsf{HCom}}$.
- Extract Alice's encoding E of her input from her request $(\mathsf{com}, sid_a, \ell, P_a, P_b, E)$ to $\mathcal{F}_{\mathsf{HCom}}$.
- Extract ℓ key pairs from $(\mathsf{transfer}, i, k^{(i)}_0, k^{(i)}_1)$ sent by Alice to $\mathcal{F}_{\mathsf{COT}}$. Send acknowledgements $\{(\mathsf{receipt}, sid_b, i, P_b, P_a) \mid i \in [\ell]\}$ on behalf of the $\mathcal{F}_{\mathsf{COT}}$ to Alice.
- Let $\mathsf{Good} = \{i \mid i \in [\ell] \text{ and } R_i \overset{?}{=} \mathsf{Extract}(k^{(i)}_0, k^{(i)}_1)\}$.
- If $|\mathsf{Good} \cap \mathsf{OpenSet}| = |\mathsf{OpenSet}|$ and $|\mathsf{Good} \cap \mathsf{EvalSet}| < \kappa$, the simulator aborts. That is, if all opened indices are Good and not enough indices in $\mathsf{EvalSet}$ are Good, the simulator aborts.
- Send input encoding E to $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$ and learn output $A \cap B$ where $A = \mathsf{Encode}^{-1}(E)$. Now, simulate Bob's final protocol message as:

$$\Big\{\mathsf{H}\big(b;\ D_b\big)\ \Big|\ b \in A \cap B\Big\} \cup \Big\{r_i\ \Big|\ i \in \{1, \ldots, |B \setminus A|\}\Big\}$$

where $r_i \xleftarrow{\$} \{0,1\}^{2\kappa}$ and

$$D_b = \{(i, \mathsf{DEval}(k_0^{(i)}, b, \mathsf{offset}_i))\}_{i \in \{\mathsf{EvalSet} \cap \mathsf{Good}\}}$$
$$\cup \{(i, \mathsf{DEval}(k_*^{(i)}, b, \mathsf{offset}_i))\}_{i \in \{\mathsf{EvalSet} \setminus \mathsf{Good}\}}$$

Through a series of hybrids we transform the real protocol execution into a simulated execution in the ideal world.

- *Hybrid 0:* We start with the actual protocol interaction where Bob participates with his input set B.
- *Hybrid 1:* We define $\mathsf{Good} = \{i \mid i \in [\ell] \text{ and } R_i \overset{?}{=} \mathsf{Extract}(k_0^{(i)}, k_1^{(i)})\}$ to be the collection of indices where we can successfully extract the set from the FSS shares. In this hybrid, we abort if $|\mathsf{Good} \cap \mathsf{OpenSet}| = |\mathsf{OpenSet}|$ and $|\mathsf{Good} \cap \mathsf{EvalSet}| < \kappa$. In Figure 6, we select parameters $|\mathsf{OpenSet}| = \ell_1$ and $|\mathsf{EvalSet}| = \ell_2$ so that abort occurs with negligible probability, making this hybrid indistinguishable from the previous hybrid.
- *Hybrid 2:* In this hybrid, we modify the message sent by Bob in step 8 of the protocol. In the previous hybrid, the message is defined as:

$$\widetilde{B} = \Big\{\mathsf{H}\big(b;\ D_b\big) \Bigm| b \in A \cap B\Big\} \cup \Big\{\mathsf{H}\big(b;\ D_b\big) \Bigm| b \in B \setminus A\Big\}$$

where $D_b = \{(i, \mathsf{DEval}(k_*^{(i)}, b, \mathsf{offset}_i)) \mid i \in \mathsf{EvalSet}\}$.
For the case $b \notin A$: if index $i \in \mathsf{Good}$ then $\mathsf{DEval}(k_0^{(i)}, b, \mathsf{offset}_i) \neq \mathsf{DEval}(k_1^{(i)}, b, \mathsf{offset}_i)$ by the correctness of the underlying ρ-dFSS scheme. When $b \notin A$ we can write

$$\mathsf{DEval}(k_*^{(i)}, a, \mathsf{offset}_i) = \mathsf{DEval}(k_0^{(i)}, b, \mathsf{offset}_i) \oplus s_i \cdot g_i$$

where g_i is a characteristic vector that indicates whether i is Good or not and s_i is Bob's choice bit. For Alice to guess $\mathsf{OPRF}(b)$ she will have to simultaneously guess Bob's choice bits for *all* indices $i \in \{\mathsf{EvalSet} \cap \mathsf{Good}\}$. From the previous hybrid, we know that $|\mathsf{EvalSet} \cap \mathsf{Good}| > \kappa$, giving Alice negligible advantage. Therefore, we replace the H outputs for $b \in B \setminus A$ with uniform outputs $r_i \xleftarrow{\$} \{0,1\}^{2\kappa}$ as shown below -

$$\widetilde{B} = \Big\{\mathsf{H}\big(b;\ D_b\big) \Bigm| b \in A \cap B\Big\} \cup \Big\{r_i \Bigm| i \in \{1, \ldots, |B \setminus A|\}\Big\}$$

where $D_b = \{(i, \mathsf{DEval}(k_*^{(i)}, b, \mathsf{offset}_i)) \mid i \in \mathsf{EvalSet}\}$.
For the case $b \in A$: if index $i \in \mathsf{Good}$ then $\mathsf{DEval}(k_0^{(i)}, b, \mathsf{offset}_i) = \mathsf{DEval}(k_1^{(i)}, b, \mathsf{offset}_i)$ by the correctness of the underlying ρ-dFSS scheme. For all indices $i \in \{\mathsf{EvalSet} \cap \mathsf{Good}\}$ Alice can express $\mathsf{DEval}(k_*^{(i)}, a, \mathsf{offset}_i) = \mathsf{DEval}(k_0^{(i)}, b, \mathsf{offset}_i)$. However, for the small set of indices $i \in \{\mathsf{EvalSet} \setminus \mathsf{Good}\}$ Alice must still guess Bob's choice bits. We reflect this change in the message sent by Bob as follows -

$$\Big\{\mathsf{H}\big(b;\ D_b\big) \Bigm| b \in A \cap B\Big\} \cup \Big\{r_i \Bigm| i \in \{1, \ldots, |B \setminus A|\}\Big\}$$

where $r_i \xleftarrow{\$} \{0,1\}^{2\kappa}$ and

$$D_b = \{(i, \mathsf{DEval}(k_0^{(i)}, b, \mathsf{offset}_i))\}_{i \in \{\mathsf{EvalSet} \cap \mathsf{Good}\}}$$
$$\cup \{(i, \mathsf{DEval}(k_*^{(i)}, b, \mathsf{offset}_i))\}_{i \in \{\mathsf{EvalSet} \setminus \mathsf{Good}\}}$$

□

Communication analysis: We suggest practical instantiations for the various components of our protocol. The $\mathcal{F}_{\mathsf{COT}}$ functionality can be realized by the committed OT construction in Jawurek et al. [JKO], in which the communication cost of choose, $\mathsf{transfer}$ and open is bounded by $\mathcal{O}(\kappa + l)$, where l is the length of the message. The $\mathcal{F}_{\mathsf{HCom}}$ functionality can be realized by the homomorphic commitment protocol of Frederiksen et al. [FJNT16], which supports additive homomorphisms over any field $\mathbb{F}$. The communication cost for each commit and open is bounded by $\mathcal{O}(t\lambda \log |\mathbb{F}|)$ and $\mathcal{O}(t \log |\mathbb{F}|)$ respectively, where t is the length of batch codes used in their construction and λ is the statistical security parameter. However, for batch invocations of commit and open their protocol achieves rate ≈ 1.

The step-by-step communication cost breakdown of our protocol is as follows:

- Step 1: Communication costs $\mathcal{O}(\kappa + l)$ for sending a commitment to the $\mathsf{OpenSet}$
- Step 2: Committing the offsets output by RShare: $\mathcal{O}(\ell \log \mathbb{G})$
- Step 3: $\mathsf{transfer}$ and choose COT cost: $\mathcal{O}(\ell\sigma)$, where σ is the FSS share size
- Step 4: decommitting $\mathsf{OpenSet}$ costs $\mathcal{O}(\kappa)$
- Step 5: For each element in OpenSet, Alice opens the key pair, and for elements not in OpenSet she opens the offsets: $\mathcal{O}(\ell(\sigma + \log |\mathbb{G}|))$
- Step 7: Communication for this step is $\mathcal{O}(|\mathsf{EvalSet}| \log |\mathbb{G}|)$
- Step 8: Bob sends a set of hash values, each of length 2κ: total cost $\mathcal{O}(|B|\kappa)$, where B is Bob's unstructured set

With cut-and-choose parameter $\ell = \mathcal{O}(\kappa)$, the total communication complexity of the protocol is $\mathcal{O}(\kappa(\sigma + \log |\mathbb{G}| + |B|))$.

5 dFSS constructions

5.1 dFSS from known bFSS constructions

Many known bFSS constructions can be easily transformed into an equivalent dFSS. In this section we present derandomizable bFSS variants for singletons (point functions), intervals, and fixed-radius ℓ_∞-balls in d dimensions, with each set including the empty set. The idea for these constructions is to make RShare output bFSS shares of a random set from the collection. The correctness and privacy of the underlying bFSS also ensures correctness and privacy of the dFSS. Further if the Share function of bFSS is **invertible** - given FSS keys k_0 and k_1 we can output the corresponding set input to Share, making the dFSS construction extractable.

The universe set $\mathcal{U} = \{0, 1, \ldots, 2^u - 1\}^d = [2^u]^d$ is parameterized by u, d. We will use the shorthand notation $\mathcal{U}_{u,d}$ to represent the same set throughout this section.

Define $\mathsf{PT}_{u,d}$ to be the family of singleton sets for the universe set $\mathcal{U}_{u,d}$ including the empty set. Define $\mathsf{INT}_{u,\delta}$ to be the family of 1-dimensional modular intervals of length δ — i.e., sets of the form $[a, b] = \{(a + i) \mod 2^u | 0 \leq i \leq \delta\}$, where $a, b \in [2^u]$ and $b = (a + \delta) \mod 2^u$, including the empty set. Define $\mathsf{BALL}_{u,d,\delta}$ to be the family of radius-δ ℓ_∞-balls in the domain $\mathcal{U}_{u,d}$, including the empty set. $\mathcal{B}_\infty(\mathsf{ctr}, \delta)$ represents a single ℓ_∞ ball of radius δ centered at ctr — *i.e.*, all points within ℓ_∞-distance δ of ctr. Similar to the collection of interval functions, the collection $\mathsf{BALL}_{u,d,\delta}$ contains ℓ_∞ balls that wrap naturally around any edge of the d-dimensional universe set. Let $\mathsf{map} : \mathcal{U}_{u,d} \to [2^{ud}]$ be any bijective mapping.

Theorem 8. *Given a strong ρ-bFSS construction for singleton sets, 1-dimensional interval and d dimensional intervals with an invertible Share function, there exists a corresponding ρ-dFSS for $\{\mathsf{PT}_{u,d}, \mathsf{INT}_{\delta,u}, \mathsf{BALL}_{\delta,u,d}\}$.*

Its proof can be found in Appendix B.1.

The best known constructions for collection of sets $\{\mathsf{PT}_{u,d}, \mathsf{INT}_{u,\delta}, \mathsf{BALL}_{u,d,\delta}\}$ [BCG+21, BGI16] have invertible Share functions - i.e. given the FSS keys, we can efficiently compute the secret shared set. For the point function construction [BGI16] for example, Eval involves computing a path in a GGM-PRF like tree, corresponding to the input x. By construction, the two parties can secret share 1 for each node in the path corresponding to some special point x^*, and other nodes secret share 0. Hence the $\mathsf{Extract}$ function can identify x^* (defining the singleton set), given both keys k_0, k_1, and by identifying the path corresponding to x^* in both trees one bit at a time.

From these previous known bFSS constructions [BCG+21, BGI16] and Theorem 8 we have the following corollaries:

Corollary 9. *There exists a 1-dFSS with pseudo-random keys for $\mathsf{PT}_{u,d}$, $\mathsf{INT}_{u,\delta}$ and $\mathsf{BALL}_{u,d,\delta}$ with share sizes $O(\kappa ud)$, $O(\kappa u)$ and $O(\kappa u^d)$ respectively.*

5.2 Generic Transformations

In this section we present some generic transformations for dFSS.

Theorem 10. *Given a 1-dFSS for a collection of sets $\mathcal{S}$ with share size σ, there exists a 1-dFSS for the collection $\overline{\mathcal{S}} = \{S | \overline{S} \in \mathcal{S}\}$ with share size σ.*

This new dFSS can be constructed, by making one of the parties output the complement of their DEval output for collection $\mathcal{S}$.

For collections $\mathcal{S}_1, \mathcal{S}_2$, we define the collection

$$\mathsf{sum}[\mathcal{S}_1, \mathcal{S}_2] = \{S_1 \triangle S_2 = (S_1 \setminus S_2) \cup (S_2 \setminus S_1) | S_1 \in \mathcal{S}_1 \text{ and } S_2 \in \mathcal{S}_2\}$$

Then we have the following theorem:

Theorem 11. *Given a 1-dFSS F_1, F_2 for a collection of sets $\mathcal{S}_1, \mathcal{S}_2$ with share size σ_1, σ_2 respectively, there exists a 1-dFSS $\mathsf{sum}[F_1, F_2]$ for the collection $\mathsf{sum}[\mathcal{S}_1, \mathcal{S}_2]$ with share size $\sigma_1 + \sigma_2$.*

Further note that, if F_1 and F_2 correspond to dFSS for disjoint sets S_1 and S_2 respectively, then $\mathsf{sum}[F_1, F_2]$ corresponds to a dFSS for the disjoint union $S_1 \cup S_2$. Further in this paper, we'll also use the following shorthand notation:

$$\mathsf{sum}[F_1, F_2, \dots, F_t] = \mathsf{sum}[F_1, \mathsf{sum}[F_2, \dots, \mathsf{sum}[F_{t-1}, F_t]]]$$

Both the above theorems follow easily from the derandomizability property of the underlying dFSS, and from the complement and sum bFSS constructions in [GRS22]. From the same construction we also have that the Eval cost of the sum dFSS would be the sum of the Eval cost of its component dFSS.

bFSS from dFSS There is a direct transformation of a dFSS scheme for a collection of sets to a bFSS scheme for the same. A formal description of the construction and the proof of the following theorem is provided in Appendix B.2.

Theorem 12. *Given a ρ-dFSS for $\mathcal{S}$ with share size σ, there exists a corresponding ρ-bFSS construction for $\mathcal{S}$ with share size σ.*

Concat Technique Given dFSS for two collections $\mathcal{S}_1, \mathcal{S}_2$, this technique gives a dFSS for their cross product $\mathcal{S}_1 \times \mathcal{S}_2$. Here the simple trick is to just concat the outputs of DEval of the two dFSS constructions. This is the same technique as the concat technique introduced in [GRS22]. See Appendix B.3 for a more detailed description of the construction and the proof of the following theorem:

Theorem 13. *Given ρ_1-dFSS F_1 for $\mathcal{S}_1$ with share size σ_1 and ρ_2-dFSS F_2 for $\mathcal{S}_2$ with share size σ_2, there exists a $(\rho_1 + \rho_2)$-dFSS $\mathsf{concat}[F_1, F_2]$ for $\mathcal{S}_1 \times \mathcal{S}_2$ with share size $\sigma_1 + \sigma_2$. Furthermore, if F_1 and F_2 dFSS have pseudo-random keys, then so does $\mathsf{concat}[F_1, F_2]$.*

Using this concat technique we can get a dFSS for an ℓ_∞ ball, which can be viewed as a cross products of d intervals.

Corollary 14. *There exists a d-dFSS for $\mathsf{BALL}_{u,d,\delta}$ with share size $O(\kappa u d)$*

Tensor Technique The point function tensor technique was first introduced by Boyle et al [BGI16] - where given an FSS for some class of functions $\mathcal{F}$ with pseudo-random keys, FSS for point functions PT, there exists an FSS for the class of functions $\mathsf{PT} \otimes \mathcal{F}$ - which contains functions of the form:

$$\mathsf{PT} \otimes \mathcal{F} = \{g_{\alpha,f} \mid f_{\alpha,1} \in PT, f \in \mathcal{F}\}$$

$$g_{\alpha,f}(x,y) = \begin{cases} f(y) & \text{if } x = \alpha \\ 0 & \text{otherwise} \end{cases}$$

In the original construction, we can replace the FSS for $\mathcal{F}$ with a dFSS for some structure $\mathcal{S}$ to get a dFSS for the following tensor structure:

$$\mathsf{PT} \otimes \mathcal{S} = \{\mathcal{U} \setminus \{\alpha\} \times S \mid \alpha \in \mathcal{U}, S \in \mathcal{S}\}$$

Theorem 15. *Given a ρ-dFSS F with pseudo-random keys for $\mathcal{S}$ with share size σ, a strong dFSS P for PT_u with share size $O(u\kappa)$, and a pseudo-random generator, there exists a ρ-dFSS $(P \otimes F)$ for the collection $\mathsf{PT} \otimes \mathcal{S}$ with share size $O(\kappa(\log|\mathcal{U}| + \sigma))$*

In the above construction, if the encoding group of $\mathsf{PT}, \mathcal{S}$ are $\mathbb{G}_{\mathsf{PT}}, \mathbb{G}_{\mathcal{S}}$ respectively, then the encoding group of $\mathsf{PT} \otimes \mathcal{S}$ is $\mathbb{G}_{\mathsf{PT}} \times \mathbb{G}_{\mathcal{S}}$. In the following subsection, we'll also assume this tensor construction DEval has an auxiliary output, where this output bit corresponds to the output of just the singleton dFSS P in the tensor construction $P \otimes F$. We'll refer to this modified eval as DEval^{aux}. From our tensor construction, we also get the following result:

Theorem 1. *Given a ρ-dFSS with pseudo-random keys for $\mathcal{S}$ with share size σ, and a pseudo-random generator, there exists a ρ-dFSS for $\overline{\mathsf{PT}} \otimes \mathcal{S} = \{\{\alpha\} \times S \mid \alpha \in \mathcal{U}, S \in \mathcal{S}\}$ with share size $O(\kappa(\log|\mathcal{U}| + \sigma))$*

This essentially follows from the fact that the tensor construction from Theorem 15 gives an auxiliary output for the strong dFSS for collection PT- allowing us to take the compliment wrt the universe $\mathcal{U}$ in the first component of the input.

The above dFSS (secret sharing $\{x\} \times S$) for input (x, y) evaluated to 0 i.e. corresponds to the element being contained in the secret shared set $\iff x = \alpha$ and $y \in S$.

5.3 An Improved spatial hashing technique

So far we've developed strong and weak dFSS schemes for a fixed radius ball in ℓ_∞ metric space. We'll use them as building blocks in this section to develop dFSS schemes for union of disjoint n balls.

Let F be a strong dFSS scheme for a single ball, then the generic transformation $\mathsf{sum}[F^n] = \mathsf{sum}[F, F, \ldots, F \text{ (n times)}]$ can be used to construct a strong dFSS for a set of n disjoint balls from the same domain. For this trivial construction, the computation cost of Eval would be n times the Eval cost of a single ball.

Improving on this construction, Garimella et al. [GRS22] introduce a *spatial hashing* technique, which gives a bFSS construction for union of n disjoint fixed radius balls in a metric space, where the Eval cost is 2^d times the Eval cost for a single ball, with the FSS key size increasing by a factor of $O(4^d)$, where d is the number of dimensions in the input domain. Assume the entire input space is divided into contiguous fixed size grid cells of side length 2δ - where δ is the radius of the input balls. In this spatial hashing construction, we prepare bFSS keys for each cell that intersects with an input ball, and pack them into an oblivious-key value storage (OKVS) [GPR+21] - giving the bFSS keys for union of balls. Here the grid coordinate is treated as the OKVS key, and the corresponding bFSS key is treated as its value. In this construction, we "shatter" each input ball, across all the 2^d intersecting grid cells - hence a bFSS key for an input ball is prepared and used for each of its neighboring cells. This blows up the size of the bFSS keys for the union of balls by a factor of 2^d. The other factor of 2^d comes from the fact that we insert $2^d n$ key-value pairs into the OKVS.

They key features of the [GRS22] spatial hashing protocol are also presented in Figure 9. It depicts the "shattering" technique we described pictorially on an example union of five l_∞ balls in 2 dimensions.

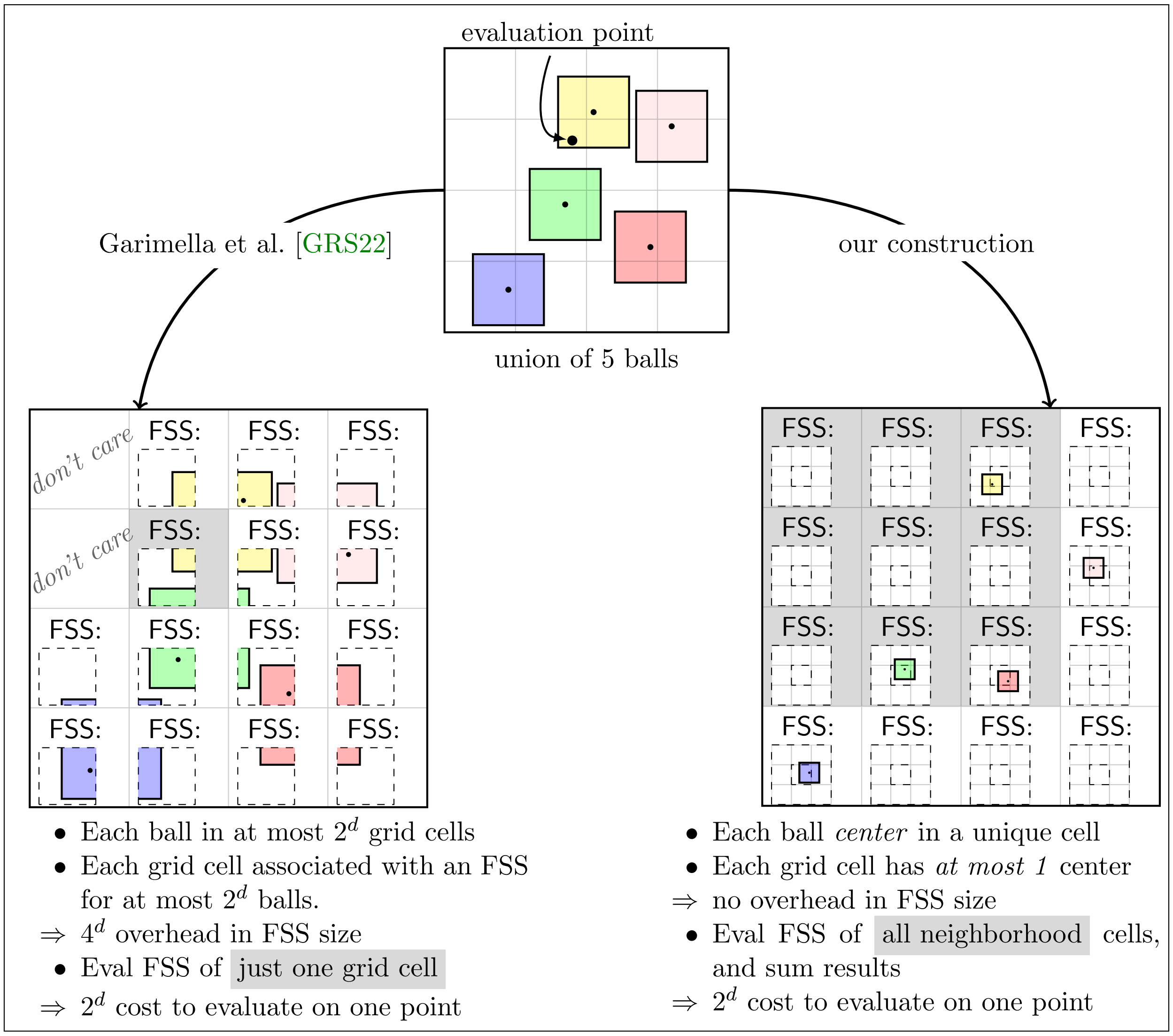
Next we'll introduce an *improved* spatial hashing construction for union of balls, which has two fold advantages over the [GRS22] construction:

- We present a dFSS instead of a bFSS scheme scheme for a union of balls. Which makes the proposed construction applicable for both the malicious and semi-honest structure-aware PSI setting.
- We avoid the use of the "shattering" technique, where each input ball is secret-shared once for each neighboring cell. This improves the FSS key size - giving it just a constant overhead compared to the trivial sum construction

High level intuition for our Spatial Hashing Construction To construct dFSS for a union of radius-δ ℓ_∞-balls, we first impose a grid structure on the input domain $\mathcal{U} = [2^u]^d$ - partitioning the input space into contiguous d dimensional grid cells, where each cell is an ℓ_∞-ball of radius δ i.e. each cell is a d dimensional cube with side length $2d$.

The dFSS key for both parties in our construction would be a cuckoo table (with 3 hash functions), where each corresponding entry of the two cuckoo tables contain dFSS keys for a single ball. In particular, we'd want the dFSS for an input ball with center ctr to be located in either of the following three locations in the cuckoo table: $H_0(\mathsf{cell}(\mathsf{ctr})), H_1(\mathsf{cell}(\mathsf{ctr}))$ or $H_2(\mathsf{cell}(\mathsf{ctr}))$, where H_i is a cuckoo hash function for $i \in \{0, 1, 2\}$ and cell function outputs the grid cell containing the input point. We refer to these grid cells containing input ball centers as *active cells.* In this construction we essentially prepare FSS keys for just the active grid cells and pack them in a cuckoo table.

To check if a point x is contained in any of the n input balls, we need to just check if the neighboring cells of x have a center of an input ball which contain x - since no input ball with center outside this neighborhood can ever contain x. Let the set of grid cells defining the neighborhood



- Each ball in at most 2^d grid cells
- Each grid cell associated with an FSS for at most 2^d balls.

$\Rightarrow$ 4^d overhead in FSS size

- Eval FSS of just one grid cell

$\Rightarrow$ 2^d cost to evaluate on one point

- Each ball *center* in a unique cell
- Each grid cell has *at most 1* center

$\Rightarrow$ no overhead in FSS size

- Eval FSS of all neighborhood cells, and sum results

$\Rightarrow$ 2^d cost to evaluate on one point

Figure 8: Illustration of the construction of Garimella et al. [GRS22] and our improved construction, for the union of n balls.

of x be $\eta(x)$. Then given a dFSS F for a single ball, we can define the Eval for a union of balls on an input point x as follows:

1. First query the cuckoo table for each grid cell in $\eta(x)$, for each of the three hash functions. This outputs $3|\eta(x)|$ FSS keys of individual balls, which can be arranged into a vector $\vec{v}$.

2. Output $\mathsf{sum}[F^{3|\eta(x)|}].\mathsf{Eval}(\vec{v}, x)$

This proposed technique of constructing FSS keys for just active cells, and evaluating only in a neighborhood is also illustrated in Figure 9 for an example input.

Compared to the trivial construction, where sum was used to take the disjoint union of n balls, in our construction above, we use the sum construction for taking the disjoint union of just the balls in the neighborhood of x. The derandomizablility of this scheme follows from the fact that the dFSS keys inserted into the cuckoo table are derandomizable. For this scheme the encoding group is $\mathbb{G}^m$, where $\mathbb{G}$ is the encoding group of a single ball (corresponding to a single entry of the cuckoo table), and m is the size of the cuckoo table.

Domain reduction optimization For the above mentioned improved spatial hashing construction, the dFSS key size would be proportional to $(\log |\mathcal{U}|)^d$ - a factor that comes from the fact that we assume each input ball is from the domain $\mathcal{U}$ and from Corollary 9. The input balls however, are of fixed radius δ - implying each input ball would be entirely contained in the neighborhood of its center. For an ℓ_∞ ball, this neighborhood is precisely a d dimensional cube of side length 6δ. We'll next improve the FSS key size for our spatial hashing construction by reducing the domain size of each input ball from the universe to just its neighborhood. This would reduce the dFSS key size, making it proportional to just $(\log(6\delta))^d$ instead of $(\log |\mathcal{U}|)^d$.

Let F be the dFSS for a single ball, and P be a dFSS for point functions. While in the above proposed spatial hashing construction we entered dFSS keys of a single ball (F) in each cuckoo entry. In our optimized spatial hashing construction, we'll put keys of dFSS $(\overline{P} \otimes F)$, where F corresponds to the dFSS for the input ball (let say at center ctr), and P corresponds to the dFSS for the point function cell(ctr). We evaluate this dFSS on input (y, x) - where x is an element of $\mathcal{U}$, and y is any cell in the neighborhood $\eta(x)$. Then from Corollary 1 this 1-dFSS outputs 0 if and only if $y = \text{cell}(\text{ctr})$ and x is contained in the ball centered at ctr. Hence, the output of this modified construction always matches that of the above construction without the domain reduction optimization as well. Furthermore, now since the dFSS F is guarded by a point function P - we can restrict the input domain of F to just a d dimensional cube of side length 6δ (containing the cells that intersect with the input ball). This is due to the fact the dFSS F evaluation matter only when you guess the cell containing the ball at center ctr correct, and otherwise the evaluation outputs 0. Note that in this optimization, we cannot use the sum construction to take the disjoint union over the neighborhood of x since the input to dFSS $(\overline{P} \otimes F)$ is not the same for each cell in the neighborhood. We illustrate this more clearly in our protocol description.

5.3.1 Formal dFSS Description

While the approach above was described for disjoint union of ℓ_∞ balls, we can extend it to other collections of union of sets as well. We'll next present some formal definitions to help define these collections of sets and our proposed dFSS construction for them.

Definition 16. *A collection of sets $\mathcal{S}$ in universe $\mathcal{U} = [2^u]^d$ is said to be a **translatable** collection if for some set $S \subset \mathcal{U}$, $\mathcal{S} = \{t + S \mid t \in \mathcal{U}\}$, where $t + S = \{t + s \mid s \in S\}$.*

Specifically note PT, INT, BALL are all translatable collections. We can define any arbitrary point of a translatable set to be a *representative element* (analogous to the "center" of a ball), so that a translated set can be specified by only the position of the representative element.

Definition 17. *For any translatable collection $\mathcal{S}$, an efficient and invertible function $\text{rep} : \mathcal{S} \to \mathcal{U}$ is said to be a representative function for $\mathcal{S}$ if $\text{rep}(t + S) = t + \text{rep}(S)$ for any $S \in \mathcal{S}, t \in \mathcal{U}$.*

As an example, for a point function/singleton set $\{x\}$ we make x the representative, and for an ℓ_∞ ball we can define its center to be a representative.

Given a collection of sets $\mathcal{S}$, we define the collection $\text{disjoint-union}_n(\mathcal{S})$ to contain a collection of n disjoint sets from the collection $\mathcal{S}$. More formally we have:

Definition 18. *For any collection of sets $\mathcal{S}$. $\text{disjoint-union}_n(\mathcal{S}) = \{\{S_i\}_{i \in [n]} \mid S_i \in \mathcal{S}, S_i \cap S_j = \emptyset \text{ for } i \neq j\}$*

We can partition the space $\mathcal{U} = [2^u]^d$ into contiguous *grid cells*, which are d-dimensional ℓ_∞-balls of radius δ. We uniquely *label* each grid cell by the point contained in it with least distance from the origin. Define a function cell_δ (parameterized by the grid size) that maps any point in the universe $\mathcal{U}$ to its unique grid cell label. Hence the function $\mathsf{cell}_\delta^{-1}$ maps a grid cell label to the set of points contained in it.

Definition 19. *For any vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_d) \in \mathcal{U}$, define the function that maps a point to its grid cell label as: $\mathsf{cell}_\delta(\boldsymbol{x}) = \lfloor \boldsymbol{x}/2\delta \rfloor = (\lfloor x_1/2\delta \rfloor, \ldots, \lfloor x_d/2\delta \rfloor)$. We also define $\mathsf{cell}_\delta^{-1}(\boldsymbol{x}) = [x_1, x_1 + 2\delta) \times [x_2, x_2 + 2\delta) \times \ldots \times [x_d, x_d + 2\delta)$, which maps any grid cell label to the set of points contained in that grid cell.*

Definition 20. *Define $G(\delta, u, d)$ = set of all grid cells = $\{\mathsf{cell}_\delta^{-1}(\mathsf{cell}(\boldsymbol{x})) \mid \boldsymbol{x} \in \mathcal{U}\}$*

We next define an active cell, as a grid cell that contains the representative element of some translatable set from the union of sets. As presented in the high level overview of the technique, we require each active cell to contain at max a single representative for a set.

Definition 21. *For some $\{S_i\}_{i\in[n]} \in \mathsf{disjoint\text{-}union}_n(\mathcal{S})$, a cell $c \in G(\delta, u, d)$ is termed **active**, if $\mathsf{rep}(S_j) \in c$ for some $j \in [n]$. Further we say the disjoint union set $\{S_i\}_{i\in[n]}$ has **unique active cells** if the number of active cells are n.*

Next we define the neighborhood $\eta(x)$ of any input x - which includes the set of all grid cells that could contain the representative of a translatable set that might contain x. We'll use this set to define the DEval function.

Definition 22. *For some translatable collection $\mathcal{S}, x \in [2^u]^d$, define $\eta(x) = \{c \in G(\delta, u, d) \mid c = \mathsf{rep}(S)$ and $x \in S$ for some $S \in \mathcal{S}\}$*

For example, for collection of sets $\mathsf{BALL}_{u,d,\delta}$, then $\eta(x)$ contains the cell $\mathsf{cell}(x)$ and each of its neighboring cells; and for collection of sets $PT_{u,d}$, $\eta(x)$ is just the singleton set containing $\mathsf{cell}(x)$.

Defining Encoding Group Let $\mathcal{S}$ be a translatable collection of sets with encoding group $\mathbb{G}_{\mathcal{S}}$, G_{PT} be the encoding group for the collection of point functions, and let grid parameter δ be set such that every set in $\mathcal{S}' = \mathsf{disjoint\text{-}union}(\mathcal{S})$ has unique active cells. Then we'll define the encoding group for $\mathcal{S}'$ as $\mathbb{G} = (\mathbb{G}_{\mathsf{PT}} \times \mathbb{G}_{\mathcal{S}})^m$, where m would correspond to the size of a 3 hash-function cuckoo table, used to store n items. Intuitively, each element of this vector encoding is a pair - where the first element encodes a grid cell, and the second element encodes the representative of the input set present at that grid cell.

The encode function Encode on input $\{S_i\}_{[n]} \in \mathsf{disjoint\text{-}union}(\mathcal{S})$ is defined as follows:

1. $X \leftarrow \{(\mathsf{cell}(\mathsf{rep}(S_i)), (\mathsf{Encode}_{\mathsf{PT}}(\mathsf{cell}(\mathsf{rep}(S_i)), \mathsf{Encode}_{\mathcal{S}}(S_i))) \,|1 \le i \le n\}$
2. **Output** cuckoo table $T = \mathsf{Cuckoo}^m_{\mathsf{H}_0,\mathsf{H}_1,\mathsf{H}_2}(X)$

By the correctness of cuckoo hashing, for any $S \in \{S_i\}_{[n]}$, its encoding is located in one of the following indexes in table T: $H_0(\mathsf{cell}(\mathsf{rep}(S)))$, $H_1(\mathsf{cell}(\mathsf{rep}(S)))$, $H_2(\mathsf{cell}(\mathsf{rep}(S)))$.

This Encode function is also efficiently invertible as follows:

$$\mathsf{Encode}^{-1}(T) = \{S \in \mathcal{S} | (c, S) \leftarrow \mathsf{Encode}_{\mathcal{S}}^{-1}(T[i]) \text{ and } c = \mathsf{cell}(\mathsf{rep}(S)), i \in [n]\} \quad (1)$$

For this encoding scheme we formally present a strong dFSS for disjoint union of a collection of translatable sets in Figure 9.

Theorem 23. *For $\mathcal{U} = [2^u]^d$, Given a strong* dFSS *for a translatable collection $\mathcal{S}$ with share size σ, and for some grid parameter δ giving unique active cells, there exists a strong* dFSS *for* disjoint-union$_n(\mathcal{S})$ *with share size $O(n(\sigma + \kappa ud))$.*

Proof.

- **Correctness**: For any input union of sets $S^* = \{S_i \in \mathcal{S} | i \in [n]\}$, and input $x \in \mathcal{U}$.
  For any $c \in \eta(x)$ and $i \in \{0, 1, 2\}$, define y_j^{idx} as the idxth party's y' value computed in DEval for the $j = (c, i)^{th}$ iteration of the for-loops. And define $y_j = y_j^0 + y_j^1$.
  Then by the correctness of the dFSS $\overline{P} \otimes F$, $y_j = 0$ iff cell c contains the representative of a translatable set in S^* that contains x. The xor of y with $y_j \oplus 1$, essentially updates the shared y value between the two parties with the complement of the y_j bit. Hence, if c does not contain the center of an input ball, or if x is not contained in the translatable set with a representative in c - then the secret shared value of y is not updated. Since all the translatable sets are disjoint, the value of secret shared y is updated from 0 to 1 **only** if x is contained in a translatable set with a representative in its neighborhood. Finally, note we xor with 1^{idx} - ensuring a complement of the final output - i.e. the 1-dFSS outputs 0 only if x is contained in a translatable set in S^*.
  if $\eta(x) = \{\eta_1, \eta_2, \ldots\}$. Then the final output secret shared DEval can also be rewritten as
  $$\overline{\overline{y_{(\eta_1,0)}} \oplus \overline{y_{(\eta_1,1)}} \oplus \overline{y_{(\eta_1,2)}} \oplus \overline{y_{(\eta_2,0)}} \oplus \overline{y_{(\eta_2,1)}} \oplus \overline{y_{(\eta_2,2)}} \oplus \cdots}$$
  Correctness follows directly from this equation as well.
- **Privacy**: Given the simulator Sim for the tensor compliment $\overline{P} \otimes F$, and a cuckoo table of size m, the simulator for disjoint-union$_n(\mathcal{S})$ outputs an m sized vector, where each of its entries is an independent output of Sim.
- **Extractability**: This follows directly from the extractability of the tensor dFSS $\overline{PT} \otimes F$.

□

Corollary 24. *There exist a 1-dFSS for union of n disjoint ℓ_∞-balls of radius δ in $\mathcal{U} = \{0, 1, \ldots, 2^u - 1\}^d$ with share size $O(n\kappa(ud + (\log \delta)^d))$*

Proof. For union of radius-δ balls, we set the grid parameter to δ to ensure unique active cells. For a universe of size 6δ, from Corollary 9 we get a strong dFSS for $\mathsf{BALL}_{6\delta,d,\delta}$ with share size $O(\kappa(\log \delta)^d)$. The corollary follows from Theorem 23. □

Weak Spatial Hashing We can further optimize the spatial hashing construction for the case where the translatable sets in the disjoint union are sufficiently "far apart". Suppose we consider a disjoint union $S^* = \{S_i \mid i \in [n]\}$ of translatable sets, such that for each $x \in \mathcal{U}$, $\eta(x)$ contains at most a single active cell. We define this collection of sets as sparse-union$(\mathcal{S})$, and its encoding group and Encode function is the same as the previous spatial hashing construction. For this collection, we'll show how to construct dFSS using just a weak dFSS for the translatable collection of sets. In particular, this construction would give us a weak dFSS for union of fixed radius ℓ_∞ balls with share size being linearly proportional to the dimension d, subject to the constraint that the distance between each pair of input balls is at least 8 times their radius.

This construction is similar to the previously proposed spatial hashing construction with some subtle differences which we list next: (as an example here we take the underlying translatable set to be a ball in some metric space)

```
Let $m$ be the size of a 3 hash $(H_0, H_1, H_2)$ cuckoo table containing $n$ elements
Given a 1-dFSS $F$ for a translatable structure $\mathcal{S}$ and a 1-dFSS $P$ for $\mathsf{PT}_{u,d}$

RShare($1^\kappa$):
  $v_0, v_1, v_R$ - size $m$ arrays
  for $i \in [m]$:
    $(v_0[i], v_1[i], v_R[i]) \leftarrow (\overline{P} \otimes F).\mathsf{RShare}(1^\kappa)$
  return $(v_0, v_1, v_R)$

DEval($1^\kappa$,idx,$v, x$, offset):
  $y \leftarrow 0$
  For $c \in \eta(x)$ :
    For $i \in \{0,1,2\}$
    $y' \leftarrow \big((\overline{P} \otimes F).\mathsf{DEval}(v[H_i(c)], (c,x) - \mathsf{offset}[H_i(c)])\big)$
  // $y'$ for both parties is a secret sharing of 0 iff $x$ is
  // contained in a translatable set located in cell $c$
    $y \leftarrow y \oplus y' \oplus 1^{\mathsf{idx}}$
  // the $1^{idx}$ xor component is used to take the FSS complement
  return $y \oplus 1^{\mathsf{idx}}$

Extract($1^\kappa, k_0, k_1$):
  Parse $k_0, k_1$ as cuckoo encodings
  Initialize empty $|k_0|$ sized vector $v_R$
  For $i \in [|k_0|]$:
    $v_R[i] \leftarrow (\overline{P} \otimes F).\mathsf{Extract}(k_0[i], k_1[i])$
    if $v_R[i] = \perp$, return $\perp$
  return $v_R$
```

Figure 9: $\mathsf{spatial\text{-}hashing}_\delta$ construction for the collection of sets $\mathsf{disjoint\text{-}union}(\mathcal{S})$ with grid size δ in domain $\mathcal{U} = \{0, 1, \ldots, 2^u - 1\}^d$

- Let F be the dFSS for a single ball, and P be a dFSS for point functions. While in the above spatial hashing construction we entered keys of dFSS $(\overline{P} \otimes F)$ in the cuckoo table, where F corresponds to the dFSS for the input ball (let say at center ctr), and P corresponds to the dFSS for the point function cell(ctr). In this construction, we'll instead enter keys for the tensor construction $(P \otimes F)$, which also has an auxiliary DEval output corresponding to the point function P.
  We evaluate this dFSS on input of the form (y, x) - where x is an element of $\mathcal{U}$, and y is any cell in the neighborhood $\eta(x)$. Then from Theorem 15 this weak dFSS outputs an all zero string if and only if $y \neq \mathsf{cell}(\mathsf{ctr})$ or x is contained in the ball centered at ctr.
- The final output of the proposed dFSS on any input x has two components (which are concatenated). The first component is 0 if and only if there exists an active cell in $\eta(x)$ - which can be computed using the auxiliary output of the $(P \otimes F)$ dFSS in each entry of the cuckoo table. The second component is an all zero string if either $\eta(x) = \emptyset$ or if x is not contained in the ball with center in its neighborhood. This second component is computed directly from the output of $(P \otimes F)$ dFSS. Hence, this weak dFSS would output an all zero string only if $\eta(x)$ is non-empty and it contains a center of a ball containing x.

The formal description of this weaker spatial hashing construction is given in Figure 10

Theorem 25. *For $\mathcal{U} = \{0, 1, \ldots, 2^u - 1\}^d$, Given ρ-dFSS for a translatable collection $\mathcal{S}$ with share size σ, and for some grid parameter δ, there exists a $(\rho + 1)$-dFSS for $\mathsf{sparse\text{-}union}_n(\mathcal{S})$ with share size $O(n(\sigma + \kappa u d))$.*

Using this theorem and the weak dFSS for $\mathsf{BALL}_{6\delta,d,\delta}$ we get a dFSS for a sparse union of ℓ infinity balls with share size that's linear in d. We define the ℓ_∞ distance between a pair of balls

```
Let $m$ be the size of a 3 hash $(H_0, H_1, H_2)$ cuckoo table containing $n$ elements
Given a $\rho$-dFSS $F$ for a translatable structure $\mathcal{S}$ and a 1-dFSS $P$ for $\mathsf{PT}_{u,d}$

RShare($1^\kappa$):
  $v_0, v_1, v_R$ - size $m$ arrays
  for $i \in [m]$:
    $(v_0[i], v_1[i], v_R[i]) \leftarrow (P \otimes F).\mathsf{RShare}(1^\kappa)$
  return $(v_0, v_1, v_R)$

Extract($1^\kappa, k_0, k_1$):
  Parse $k_0, k_1$ as cuckoo encodings
  Initialize empty $|k_0|$ sized vector $v_R$
  For $i \in [|k_0|]$:
    $v_R[i] \leftarrow (P \otimes F).\mathsf{Extract}(k_0[i], k_1[i])$
    if $v_R[i] = \perp$, return $\perp$
  return $v_R$

DEval($1^\kappa$,idx,$v, x$,offset):
  $y \leftarrow (0, 0)$
  For $c \in \eta(x)$ :
    For $i \in \{0, 1, 2\}$
    $(y_0, y_1) \leftarrow ((P \otimes F).\mathsf{DEval}^{aux}(v[H_i(c)], (c, x) - \mathsf{offset}[H_i(c)]))$
   // here $y_0$ corresponds to the auxiliary output
    $y \leftarrow y \oplus (y_0, y_1)$
  return $(y[0] \oplus 1^{\mathsf{idx}})||y[1]$
```

Figure 10: Weak spatial hashing construction for the collection of sets $\mathsf{sparse\text{-}union}(\mathcal{S})$ with grid size δ in domain $\mathcal{U} = \{0, 1, \dots, 2^u - 1\}^d$

in $\mathsf{BALL}_{6\delta,d,\delta}$ to be the distance between their centers. Then we have the following corollary:

Corollary 26. *There exists a $(d+1)$-dFSS for the union of n, ℓ_∞-balls of radius δ, with pairwise distance $> 8\delta$, in $\mathcal{U} = \{0, 1, \dots, 2^u - 1\}^d$, with share size $O(\kappa n u d)$.*

5.4 Discussion on co-domain of Encode^{-1}

Given a dFSS for collection of sets $\mathcal{S}$ with encode function $\mathsf{Encode} : \mathcal{S} \to \mathbb{G}$. For each $S \in \mathcal{S}$, we have $\mathsf{Encode}^{-1}(\mathsf{Encode}(S)) = S$. In our PSI protocol however, a malicious Alice may commit offsets corresponding to some element in the encoding space $g \in \mathbb{G}$. Note that for $\mathcal{S} \in \{\mathsf{PT}_{u,d}, \mathsf{INT}_{d,\delta}, \mathsf{BALL}_{u,d,\delta}\}$, for any $g \in \mathbb{G}$, $\mathsf{Encode}^{-1}(g) \in \mathcal{S}$ — *i.e.*, Encode is a bijection for these collections. Hence for $\mathcal{S} \in \{\mathsf{PT}_{u,d}, \mathsf{INT}_{d,\delta}, \mathsf{BALL}_{u,d,\delta}\}$ the corrupt party cannot input a set outside the expected collection.

For union of n disjoint ℓ_∞ balls with encoding function $\mathsf{Encode} : \mathsf{disjoint\text{-}union}_n(\mathsf{BALL}_{u,d,\delta}) \to \mathbb{G}$ and for any $g \in G$, from Equation 1, we have that $\mathsf{Encode}^{-1}(g)$ may contain at most one input ball for each entry in the cuckoo table. Hence a corrupt party may be able to input a set of at most m balls in our PSI protocol, where m is the size of the cuckoo table to store n items.

References

[BBCG+21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 762–776. IEEE, 2021.

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

[BCG$^+$19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Heidelberg, August 2019.

[BCG$^+$21] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 871–900. Springer, 2021.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

[Bea95] Donald Beaver. Precomputing oblivious transfer. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 97–109. Springer, Heidelberg, August 1995.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 337–367. Springer, 2015.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, 2016.

[BGI19] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2019.

[BGIK22] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I Kolobov. Programmable distributed point functions. *Cryptology ePrint Archive*, 2022.

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

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

[dCP22] Leo de Castro and Anitgoni Polychroniadou. Lightweight, maliciously secure verifiable function secret sharing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 150–179. Springer, 2022.

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

[DKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 213–231. Springer, Heidelberg, December 2010.

[DMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 125–142. Springer, Heidelberg, June 2009.

[DMRY11] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Secure efficient multiparty computing of multivariate polynomials and applications. In Javier Lopez and Gene Tsudik, editors, *ACNS 11*, volume 6715 of *LNCS*, pages 130–146. Springer, Heidelberg, June 2011.

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

[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. Springer, Heidelberg, February 2005.

[FJNT16] Tore Kasper Frederiksen, Thomas P Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. On the complexity of additively homomorphic uc commitments. In *Theory of Cryptography Conference*, pages 542–565. Springer, 2016.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, Heidelberg, May 2004.

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

[GN19] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 154–185. Springer, Heidelberg, May 2019.

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

[GRS22] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. Structure-aware private set intersection, with applications to fuzzy matching. In *CRYPTO: Advances in Cryptology*, 2022.

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

[HFH99a] Bernardo A. Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *ACM CONFERENCE ON ELECTRONIC COMMERCE*. ACM, 1999.

[HFH99b] Bernardo A Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 78–86, 1999.

[HL08] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 155–175. Springer, Heidelberg, March 2008.

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

[JKO] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits.

[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. Springer, Heidelberg, September 2010.

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

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 241–257. Springer, Heidelberg, August 2005.

[Mea86] Catherine Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *1986 IEEE Symposium on Security and Privacy*, pages 134–134. IEEE, 1986.

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

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

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

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

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

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 797–812, 2014.

[RR17a] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 235–259. Springer, Heidelberg, April / May 2017.

[RR17b] 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. ACM Press, October / November 2017.

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

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-OLE: Improved constructions and implementation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1055–1072. ACM Press, November 2019.

Appendix

A Weak Boolean FSS definition

Here we present the original weak boolean FSS formulation from [GRS22] with no false positives.

Definition 27 (bFSS syntax). *Let $\mathcal{S} \subseteq 2^{\mathcal{U}}$ denote a family of sets over input domain $\mathcal{U}$, security parameter κ. A 2-party ρ-bFSS scheme with algorithms (Share,Eval) has the following syntax:*

- $(k_0, k_1) \leftarrow \mathsf{Share}(1^\kappa, S)$*: is an algorithm with input the security parameter and (the description of) a set $S \in \mathcal{S}$ and it outputs key shares (k_0, k_1).*
- $y_{\mathsf{idx}} \leftarrow \mathsf{Eval}(1^\kappa, \mathsf{idx}, k_{\mathsf{idx}}, x)$*: is a deterministic algorithm with input security parameter κ, party index $\mathsf{idx} \in \{0,1\}$, key share k_{idx} and any $x \in \mathcal{U}$. It outputs a binary string y_{idx} of length ρ.*

Definition 28 (bFSS security). *A 2-party ρ-bFSS scheme $(\mathsf{Share}, \mathsf{Eval})$ for $\mathcal{S} \subseteq 2^{\mathcal{U}}$ is* **secure** *if is satisfies the following conditions:*

Given a bFSS F for singleton set with $\mathcal{U} = [2^{ud} + 1]$

```
RShare(1^κ):
    Pick random encoding $r \leftarrow_R \mathcal{U}$
    $(k_0, k_1) \leftarrow F.\mathsf{Share}(1^\kappa, a)$
    return $(k_0, k_1, r)$

DEval(1^κ, idx, k, offset, x):
    $y_{\mathsf{idx}} \leftarrow F.\mathsf{Eval}(1^\kappa, \mathsf{idx}, k, \mathsf{map}(x) - \mathsf{offset})$
    return $y_{\mathsf{idx}}$
```

Figure 11: (RShare,DEval) functions of a dFSS for $\mathsf{PT}_{u,d}$

- ***Correctness for yes-instances:*** *For every $S \in \mathcal{S}$, $x \in S$:*

$$\Pr\left(y_0 \oplus y_1 = 0^\rho \;\middle|\; \begin{array}{l} (k_0, k_1) \leftarrow \mathit{Share}(1^\kappa, S) \\ y_0 \leftarrow \mathit{Eval}(1^\kappa, 0, k_0, x) \\ y_1 \leftarrow \mathit{Eval}(1^\kappa, 1, k_1, x) \end{array} \right) = 1$$

- ***Correctness for no-instances:*** *For every set $S \in \mathcal{S}$, $x \in \mathcal{U} \setminus S$:*

$$\Pr\left(y_0 \oplus y_1 \neq 0^\rho \;\middle|\; \begin{array}{l} (k_0, k_1) \leftarrow \mathit{Share}(1^\kappa, S) \\ y_0 \leftarrow \mathit{Eval}(1^\kappa, 0, k_0, x) \\ y_1 \leftarrow \mathit{Eval}(1^\kappa, 1, k_1, x) \end{array} \right) = 1$$

- ***Privacy:*** *For every set $S \in \mathcal{S}$ and index $\mathit{idx} \in \{0, 1\}$ there exists a simulator Sim such that the following distributions are computationally indistinguishable in the security parameter:*

$$\boxed{\begin{array}{l} (k_0, k_1) \leftarrow \mathsf{Share}(1^\kappa, S) \\ \text{return } k_{\mathsf{idx}} \end{array}} \cong_\kappa \mathit{Sim}(1^\kappa, \mathit{idx})$$

B dFSS proofs/constructions

B.1 dFSS for singleton sets, intervals and a d-dimensional ball

The construction for the following theorem can be found in Figure 11.

Theorem 8. *Given a strong ρ-bFSS construction for singleton sets, 1-dimensional interval and d dimensional intervals with an invertible Share function, there exists a corresponding ρ-dFSS for $\{\mathit{PT}_{u,d}, \mathit{INT}_{\delta,u}, \mathit{BALL}_{\delta,u,d}\}$.*

Proof. We'll show how this theorem holds for the class of point functions, but a similar proof would follow for the other two collections as well.

To encode collection of point functions, we define its encoding group $\mathbb{G}_{\mathsf{PT}}$ as $[2^{ud} + 1]$. The group element 2^{ud} would encode the null set. Which helps us define the Encode function as follows:

$$\mathsf{Encode}_{\mathsf{PT}}(\{x\}) = \mathsf{map}(x) \text{ for } x \in \mathcal{U}_{u,d} \text{ and } \mathsf{Encode}_{\mathsf{PT}}(\emptyset) = 2^{ud}$$

Furthermore, we can define $\mathsf{Encode}_{\mathsf{PT}}^{-1}$ using the inverse of map. The proposed construction is presented in Figure 11. Here the RShare outputs keys for a random singleton set, which is later derandomized in DEval to a chosen element by using an appropriate offset. Note, here we can encode a null set as well, by making the offset derandomize the singleton set to $\{2^{ud}\}$ - which is outside the domain of the dFSS. Next we show how each of the dFSS properties are satisfied.

```
Given a dFSS $F$ for some $\mathcal{S}$ with encode function Encode

Share($1^\kappa, S \in \mathcal{S}$):
    $(k_0, k_1, R) \leftarrow F.\mathsf{RShare}(1^\kappa)$
    offset = Encode($S$) $- R$
    return $((k_0, \mathsf{offset}), (k_1, \mathsf{offset}))$

Eval($1^\kappa, \mathsf{idx}, key = (k_{\mathsf{idx}}, \mathsf{offset}), x$):
    return $F.\mathsf{DEval}(1^\kappa, \mathsf{idx}, k_{\mathsf{idx}}, \mathsf{offset}, x)$
```

Figure 12: (Share,Eval) of a bFSS for a collection of sets given a dFSS F for the same structure

- **Correctness**: For any $x, x^* \in PT_{u,d}$, and offset $\mathsf{offset} = \mathsf{Encode}_{\mathsf{PT}}(x^*) - r$, define y_{idx} to be $F.\mathsf{Eval}(1^\kappa, \mathsf{idx}, k, \mathsf{map}(x) - \mathsf{offset})$. Then $y_0 \oplus y_1 = 1 \iff \mathsf{map}(x) = \mathsf{map}(x^*)$ by the correctness of the underlying bFSS scheme, and else $y_0 \oplus y_1 = 1$.
  If the underlying bFSS has a correctness error - in the sense that the keys k_0, k_1 evaluate to a null set with negligible probability, then the dFSS would give incorrect output for the element x^* if its not $\emptyset$. This leads to a negligible error for a *yes-instance* of this dFSS. This negligible error does arise in point function constructions in [BCG+21, BGI16], which we'll assume in this work.
  For a *no-instance* of this dFSS, there is never any error in the output. Since the output $y_0 \oplus y_1$ is always zero when input $x \neq x^*$.
- **Security**: This follows directly from the privacy of the underlying bFSS, and by the fact that r acts as a one-time in the offset $\mathsf{Encode}(x^*) - r$.
- **Extractability**: The Extract functions outputs the secret shared singleton element r, given the two keys k_0, k_1. It can be constructed using the invertible Share function as follows:

$$\mathsf{Extract}(k_0, k_1) = \mathsf{Encode}(\mathsf{Share}^{-1}(1^\kappa, k_0, k_1))$$

Similarly a bFSS for 1 dimensional interval and d dimensional fixed radius ℓ_∞ can give us a corresponding dFSS by "reducing the domain size" to ensure a null set can be encoded by a set outside the specified domain of dFSS.

□

B.2 bFSS from dFSS

Theorem 12. *Given a ρ-dFSS for $\mathcal{S}$ with share size σ, there exists a corresponding ρ-bFSS construction for $\mathcal{S}$ with share size σ.*

Proof. This construction is presented in Figure 12.

- **Correctness**: The term $\sum_{\mathsf{idx}\in\{0,1\}} \mathsf{Eval}(\mathsf{idx}, (k_{\mathsf{idx}}, \mathsf{Encode}(S)-R), x) = \sum_{\mathsf{idx}\in\{0,1\}} \mathsf{Eval}(\mathsf{idx}, k_{\mathsf{idx}}, \mathsf{Encode}(S)-R, x)$. Hence this follows directly from the correctness definition of dFSS.
- **Security**: The simulator for the bFSS simply runs the simulator for the dFSS.

□

B.3 concat

Theorem 13. *Given ρ_1-dFSS F_1 for $\mathcal{S}_1$ with share size σ_1 and ρ_2-dFSS F_2 for $\mathcal{S}_2$ with share size σ_2, there exists a $\rho_1 + \rho_2$-dFSS $\mathsf{concat}[F_1, F_2]$ for $\mathcal{S}_1 \times \mathcal{S}_2$ with share size $\sigma_1 + \sigma_2$. Furthermore, if F_1 and F_2 dFSS have pseduo-random keys, then so does $\mathsf{concat}[F_1, F_2]$.*

```
RShare(1^κ):
    Initialize $k_0, k_1, R$ as empty associated arrays
    $(k_0[0], k_1[0], R[0]) \leftarrow F_1.\mathsf{RShare}(1^\kappa)$
    $(k_0[1], k_1[1], R[1]) \leftarrow F_2.\mathsf{RShare}(1^\kappa)$
    return $(k_0, k_1, R)$

DEval(1^κ,idx,k_idx, offset, x):
    $y_1 \leftarrow F_1.\mathsf{DEval}(1^\kappa, \mathsf{idx}, k_{\mathsf{idx}}[0], \mathsf{offset}[0], x)$
    $y_2 \leftarrow F_2.\mathsf{DEval}(1^\kappa, \mathsf{idx}, k_{\mathsf{idx}}[1], \mathsf{offset}[1], x)$
    return $y_1 || y_2$

Extract(1^κ, k_0, k_1):
    return $(F_1.\mathsf{Extract}(1^\kappa, k_0[0], k_1[0]),$
    $F_2.\mathsf{Extract}(1^\kappa, k_0[1], k_1[1]))$
```

Figure 13: dFSS for cross product $S_1 \times S_2$ given dFSS for S_1 and S_2

Proof. The construction is presented in Figure 13. Given encode functions $\mathsf{Encode}_1 : \mathcal{S}_1 \to \mathbb{G}_1$ and $\mathsf{Encode}_2 : \mathcal{S}_2 \to \mathbb{G}_2$ for dFSS F_1 and F_2 respectively, we define encode function for concat $[F_1, F_2]$ $\mathsf{Encode} : \mathcal{S}_1 \times \mathcal{S}_2 \to \mathbb{G}_1 \times \mathbb{G}_2$ as:

$$\mathsf{Encode}(S_1, S_2) = (\mathsf{Encode}_1(S_1), \mathsf{Encode}_2(S_2))$$

Note that this Encode function is efficiently invertible if the two component Encode functions are. Encodings with respect to the new Encode function form a group which is a direct product of the groups for the component FSS.

- **Correctness**: The correctness of $F = \mathsf{concat}[F_1, F_2]$ reduces to the correctness of F_1 and F_2. The first k_1 bits of the term $y = \sum_{\mathsf{idx}\in\{0,1\}} F.\mathsf{DEval}(\mathsf{idx}, (k_{\mathsf{idx}}, \mathsf{Encode}(S_1, S_2) - (R_1, R_2)), (x, y))$ equal $\sum_{\mathsf{idx}\in\{0,1\}} F_1.\mathsf{Eval}(\mathsf{idx}, (k_{\mathsf{idx}}, \mathsf{Encode}(S_1) - R_1), x)$ and the last k_2 bit of y equal $\sum_{\mathsf{idx}\in\{0,1\}}$ $F_2.\mathsf{Eval}(\mathsf{idx}, (k_{\mathsf{idx}}, \mathsf{Encode}(S_2) - R_2), x)$. Hence, $y = 0^{k_1+k_2}$ for $(x, y) \in S_1 \times S_2$. And if $x \notin S_1$, the first k_1 bits are not all zero bits with probability at least p_1, and if $x \notin S_2$, the last k_2 bits are not all zero bits with probability at least p_2.
- **Security**: The simulator Sim for $\mathsf{concat}[F_1, F_2]$ invokes the simulators for F_1 and F_2, which output (k_0, offset_0) and (k_1, offset_1). Then Sim outputs $((k_0, k_1), (\mathsf{offset}_0, \mathsf{offset}_1))$.
- **Extractability**: This is reduced to the extractability property of the two underlying dFSS F_1 and F_2 as shown in Figure 13.

□