New Framework for Structure-Aware PSI From Distributed Function Secret Sharing

Dung Bui
IRIF, Université Paris Cité
bui@irif.fr

Gayathri Garimella
Brown University
gayathri_garimella@brown.edu

Peihan Miao
Brown University
peihan_miao@brown.edu

Phuoc Van Long Pham
Brown University
phuoc_pham_van_long@brown.edu

Abstract

Private set intersection (PSI) allows two parties to jointly compute the intersection of their private sets without revealing any additional information. Structure-aware PSI (sa-PSI), introduced by Garimella et al. (Crypto'22), is a variant where Alice's input set has a publicly known structure and Bob's input set remains unstructured, enabling new applications like fuzzy PSI. Their construction relies solely on lightweight cryptographic primitives such as symmetric-key primitives and oblivious transfer (OT) extension. Since then, there has been active research on sa-PSI based on lightweight cryptography. Notably, recent work by Garimella et al. (Crypto'24) achieves sa-PSI with both communication and computation costs only scaling with the description size of Alice's set, rather than its potentially large cardinality. However, this line of work remains largely theoretical, lacking efficient concrete implementations.

In this work, we close this gap by presenting a new framework for sa-PSI that achieves practical efficiency. We identify and eliminate a hidden multiplicative overhead proportional to the security parameter (e.g., 128) in prior symmetric-key-based sa-PSI constructions. A key building block of our new framework is a distributed Function Secret Sharing (dFSS) key generation protocol tailored to the structure of Alice's set, which may be of independent interest. To demonstrate the practicality of our framework, we extend our dFSS protocol to support incremental evaluation, introduce new techniques for spatial hashing, and develop several new optimization techniques, including reducing the exponential dependence on dimension and enabling load balancing between the two parties.

We instantiate our framework for structured sets defined by unions of d-dimensional ℓ_∞ balls, and implement our protocols using only lightweight symmetric-key primitives and OT extension. Our experiments show concrete performance improvements of up to $27\times$ speedup in computation and $7.7\times$ reduction in communication in low-dimensional, large-radius settings compared to existing public-key-based fuzzy PSI protocols by van Baarsen & Pu (Eurocrypt'24) and Gao et al. (Asiacrypt'24).

Contents

1 Introduction

Private set intersection (PSI) enables two parties, each holding a private set of elements, to jointly compute the intersection of their sets without revealing any additional information than the intersection itself. Due to its wide range of applications such as password breach detection [CHLR18, CMdG+21], mobile private contact discovery [KRS+19], and online advertising measurement [IKN+19], tremendous effort has been made towards achieving practically efficient PSI over the past decades. Today, PSI protocols are extremely efficient, leveraging lightweight cryptographic primitives and taking less than a second to compute the intersection of two sets each consisting of a million elements [RR22]. In response to diverse and increasingly complex practical needs, there has been growing interest in PSI variants, including PSI Cardinality (PSI-card) [DPT20, WY23], which outputs only the size of the intersection; Circuit PSI [HEK12, PSTY19], which outputs secret shares of the elements in the intersection; Labeled PSI (LPSI) [CHLR18, CMdG+21], which reveals to the receiver the labels associated with elements in the intersection; Threshold PSI [GS19, BMRR21, LZQ23], which reveals the intersection only if its size is greater than a specified threshold; and many more.

Recent work by Garimella et al. [GRS22] introduced the notion of *structure-aware PSI (sa-PSI)*, where Alice's input set has some publicly known structure and Bob's input set remains unstructured. For instance, Alice's set S_A consists of a union of high-dimensional balls with a radius δ, while Bob's input set S_B is an unstructured set of points. sa-PSI can be viewed as a generalization of fuzzy PSI, which aims to identify approximate rather than exact matches. Specifically, suppose Alice and Bob each have a set of points A and B, respectively, and they want to identify fuzzy matches, namely $(x, y) \in A \times B$ satisfying $D(x, y) \leq \delta$, where D is a public distance metric (e.g. Hamming, Minkowski (ℓ_p), or ℓ_∞) and δ is a public threshold. This problem can be solved using sa-PSI by having Alice expand her set to include nearby points within distance δ, namely $S_A := \{x' | x \in A \wedge D(x', x) \leq \delta\}$. This abstraction is particularly useful in applications such as biometric authentication, where the goal is simply to determine whether a given fingerprint (or face) approximately matches one in a database [UCK+21]. Another application is privacy-preserving proximity testing [HOS17], where two parties want to determine whether they are sufficiently close to each other without revealing their precise locations or relative distance.

Naïvely applying standard PSI to this problem would result in computation and/or communication costs scaling with the size of S_A, which can be prohibitively large. However, although S_A may be much larger than S_B, it has a succinct *description size*. sa-PSI protocols [GRS22, GRS23] allow Alice to learn the intersection $S_A \cap S_B$ with communication cost that only scales with the description size of S_A rather than its cardinality. In more recent work, Garimella et al. [GGM24] achieves *both* computation and communication complexity that scale with the description size of S_A. Moreover, these constructions only rely on lightweight cryptographic tools, including oblivious transfer (OT) extension [IKNP03] and symmetric-key-based function secret sharing (FSS) [BGI15]. However, this line of work remains largely theoretical, and has not yet demonstrated practical, efficient implementations.

In contrast, another line of work [vBP24, GQL+24] presents concretely efficient protocols for fuzzy PSI, albeit relying on computationally more expensive public-key primitives such as additively homomorphic encryption (AHE) and assumptions like Diffie-Hellman (DDH).

The PSI literature has demonstrated the advantages of symmetric-key primitives. The most efficient PSI protocols [KKRT16, RS21, RR22] rely on lightweight cryptographic techniques such as

OT extension [IKNP03] or vector oblivious linear evaluation (VOLE) [BCGI18, BCG+19], rather than computationally more expensive public-key-based approaches. However, this efficiency gap has not yet shown up in the context of structure-aware/fuzzy PSI.

In this work, we bridge this gap by developing concretely efficient, symmetric-key-based sa-PSI, outperforming all public-key-based constructions. When comparing the symmetric-key-based sa-PSI [GRS22, GRS23, GGM24] with public-key-based fuzzy PSI [vBP24, GQL+24], we observe that the sa-PSI framework inherently incurs an *additional* multiplicative factor of κ (e.g., 128), the computational security parameter, in both computation and communication costs, which was hidden in its asymptotic analysis in prior works. This factor significantly impacts the practical efficiency of the symmetric-key-based approaches. To overcome this, we introduce a new framework for sa-PSI that eliminates this multiplicative overhead, resulting in a concretely efficient protocol.

1.1 Our Contributions

New framework for sa-PSI from dFSS. We present a new framework for structure-aware PSI based on lightweight cryptography that eliminates the aforementioned multiplicative factor of κ in both communication and communication costs in prior works. The key building block of our new framework is a *distributed Function Secret Sharing (dFSS)* key generation protocol tailored to the corresponding set family. At a high level, instead of having Alice generate both FSS key shares and repeat κ times to achieve κ-bit security, our new framework has both parties jointly perform the FSS key generation such that each party obtains only one share, hence removing the need for repetition.

Construction of (incremental) dFSS for d-dimensional ℓ_∞ balls. We instantiate our new framework for structured sets defined as unions of d-dimensional balls under the ℓ_∞ distance metric. In particular, we present a dFSS key generation protocol for this set family, which may be of independent interest. To ensure that both computation and communication scale only with the description size of the structured set, we further extend our protocol to support *incremental* FSS evaluation.

Building compatibility with spatial hashing. We develop new techniques to support the spatial hashing optimization [GRS22, GRS23]. The previous framework requires Alice to generate oblivious key-values stores (OKVS) [GPR+21] to encode FSS key shares, which is incompatible with our new framework that performs FSS key generation in a distributed manner. To resolve this issue, we introduce new techniques based on cuckoo hashing [Pag08] and oblivious pseudorandom functions (OPRFs) [FIPR05].

Additional optimizations. We propose several additional optimizations that substantially improve concrete efficiency, including a half-tree optimization for the dFSS construction, methods to reduce the exponential dependence on the dimension, and load-balancing between the two parties.

Implementation. We provide the first implementation for sa-PSI from FSS, and show concrete efficiency improvements over public-key-based fuzzy PSI protocols [vBP24, GQL+24]. All our building blocks only rely on lightweight symmetric-key cryptographic operations (such as pseudorandom generators and hash functions) and OT extension.

1.2 Related Work

Private Set Intersection (PSI). Standard PSI protocols have seen significant performance improvements in recent years [HFH99, FNP04, PSSW09, KKRT16, PRTY19, CM20, RS21], particularly with the development of pseudorandom correlation generators [BCG+19, BCG+20], which facilitated faster and more scalable protocols [RS21, RR22, CILO22, BC23]. These protocols typically exhibit linear complexity in both communication and computation relative to the input size. However, when extending to fuzzy PSI, where two elements are considered matching if they are "close" in a certain distance metric, this linearity becomes problematic. Specifically, Alice's input effectively expands to size $N \cdot \delta^d$, where N is the number of elements (or balls) in her set, δ is the ball's diameter, and d is the dimension, resulting in high overhead.

Structured-Aware PSI. Garimella, Rosulek, and Singh [GRS22] introduced a line of PSI protocols for structured sets, in which Alice (the receiver) holds N balls (each with diameter δ in d-dimensional space) and Bob (the sender) holds M unstructured points. Alice learns which of Bob's points lie inside any of her balls. Their solution employs weak function secret sharing (FSS) [BGI15] and oblivious transfer (OT). The follow-up work of [GRS23] achieves malicious security via a cut-and-choose technique ensuring consistent encoding across multiple FSS shares. Both works achieve communication cost that only scales with the description size of Alice's structured set. However, they require Alice's computation to scale with the cardinality of her set and exponentially with with description size. To address this, Garimella et al. [GGM24] introduced incremental FSS that is compatible with a succinct description of Alice's input, reducing her computational overhead. This line of work relies solely on lightweight cryptographic tools including pseudorandom generators, hash functions, and OT extension [IKNP03]. However, these results remain primarily theoretical and lack concrete implementations.

In a separate line of work, van Baarsen and Pu [vBP24] proposed a fuzzy PSI protocol based on ElGamal encryption [ElG85] under the Decisional Diffie-Hellman (DDH) assumption. In addition to the ℓ_∞ distance metric, their construction also supports Minkowski distance (ℓ_p for $p \in [1, \infty)$) and Hamming distance. Most recently, Gao et al. [GQL+24] built upon this work and introduced a new primitive called fuzzy mapping (Fmap), where elements are mapped to identifier sets if they are close. Their construction achieves strictly linear complexity in the size of the inputs and is built using additive homomorphic encryption (AHE), also under the DDH assumption. A recent work by Richardson et al. [RRX24] proposed a fuzzy PSI construction that avoids the reliance on public-key assumptions, but their construction is based on generic MPC approach, i.e., garbling circuits.

For fuzzy PSI with Hamming distance, Uzun et al. [CFR23] proposed a construction based on fully homomorphic encryption (FHE) [Gen09]. Chakraborti et al. [CFR23] later presented a more efficient protocol that leverages Oblivious Linear Evaluation (OLE) in combination with AHE. These protocols perform a brute-force search over all $N \cdot M$ pairs of inputs from both parties, which leads to a blowup in communication and computation complexity. A more recent work by Chongchitmate et al. [CLO24] reduces this overhead for structured data by assuming that elements are either "close" or sufficiently "far apart".

Function Secret Sharing. Function Secret Sharing (FSS), introduced by Boyle et al. [BGI15], enables compactly secret sharing a function. This primitive has since been extended and optimized for various classes of functions, including point functions, multi-point functions, comparison functions, and d-dimensional interval functions [BGI16, BCG+21, BGIK22, BCG+22]. Boneh et al. [BBC+21]

introduced a variant of FSS for comparison functions with a new property called "incremental," which allows evaluations on arbitrary prefixes of a string.

In the context of structure-aware PSI, Garimella et al. [GRS22] introduced the concept of Boolean FSS (bFSS), extending the original FSS framework [BGI15] to succinctly represent structured sets. They also proposed constructions for computing the union of disjoint structured sets. Building on this, Garimella et al. [GGM24] formalized Incremental Boolean FSS (ibFSS) and gave a construction for d-dimensional ℓ_∞ balls, relying on an incremental distributed comparison function (iDCF). In this work, we extend the notion of iDCF and present a new, efficient construction that supports distributed key generation using pseudorandom generators and OT. This contribution may be of independent interest.

2 Technical Overview

We consider structure-aware private set intersection (sa-PSI) where Alice holds a structured input set S_A and Bob holds an unstructured input set S_B.

Prior framework for sa-PSI from FSS. We begin by reviewing the framework introduced in prior works [GRS22, GRS23, GGM24] for constructing sa-PSI from FSS. The core component is an FSS scheme tailored to the *set membership function (SMF)* of Alice's structured input set S_A, that is,

$$\mathsf{SMF}_{S_A}(\vec{y}) = \begin{cases} 0 & \text{if } \vec{y} \in S_A \\ 1 & \text{otherwise} \end{cases} .$$

Specifically, the FSS scheme includes two algorithms $(\mathsf{Share}, \mathsf{Eval})$ where $(k_0, k_1) \leftarrow \mathsf{Share}(1^\kappa, S_A)$ succinctly secret-shares the function SMF_{S_A} by generating a pair of keys (k_0, k_1) with the properties that each key share k_b hides the structure S_A, and that $\mathsf{Eval}(k_0, \vec{y}) \oplus \mathsf{Eval}(k_1, \vec{y}) = 0$ if $\vec{y} \in S_A$ and $\mathsf{Eval}(k_0, \vec{y}) \oplus \mathsf{Eval}(k_1, \vec{y}) = 1$ otherwise.[1]

The sa-PSI framework is constructed from FSS as follows. Alice generates ℓ *independent* FSS shares of her set S_A by computing ℓ key pairs $(k_0^{(i)}, k_1^{(i)}) \leftarrow \mathsf{Share}(1^\kappa, S_A)$ for each $i \in [\ell]$. Bob samples a random string $s \leftarrow \{0,1\}^\ell$. The parties then invoke ℓ instances of oblivious transfer (OT) [Rab05], where Alice as sender inputs $\{(k_0^{(i)}, k_1^{(i)})\}_{i\in[\ell]}$ and Bob as receiver inputs $\{s[i]\}_{i\in[\ell]}$, allowing Bob to receive $k_{s[i]}^{(i)}$ for each $i \in [\ell]$, while keeping $s[i]$ hidden from Alice.

Bob then defines the following function:

$$\mathsf{F}(\vec{y}) = \mathsf{H}\left(\mathsf{Eval}(k_{s[1]}^{(1)}, \vec{y}) \| \ldots \| \mathsf{Eval}(k_{s[\ell]}^{(\ell)}, \vec{y})\right),$$

and sends the set $Y := \{\mathsf{F}(\vec{y}) \mid \vec{y} \in S_B\}$ to Alice.

On Alice's side, she can only compute this function for $\vec{y} \in S_A$; otherwise the output looks pseudorandom to her. Notice that for all points $\vec{y} \in S_A$, both FSS shares evaluate to the same output, namely, $\mathsf{Eval}(k_0^{(i)}, \vec{y}) = \mathsf{Eval}(k_1^{(i)}, \vec{y})$ for all $i \in [\ell]$. Hence, the function evaluation $\mathsf{F}(\vec{y})$ is independent of Bob's choice bits, and can be computed by Alice:

$$\mathsf{F}(\vec{y}) = \mathsf{H}\left(\mathsf{Eval}(k_0^{(1)}, \vec{y}) \| \ldots \| \mathsf{Eval}(k_0^{(\ell)}, \vec{y})\right).$$

[1] For ease of explanation, we illustrate the key ideas using FSS with single-bit outputs. However, it suffices to consider *weak* FSS where the evaluation functions output t-bit strings such that $\mathsf{Eval}(k_0, \vec{y}) \oplus \mathsf{Eval}(k_1, \vec{y}) = 0^t$ if $\vec{y} \in S_A$ and $\mathsf{Eval}(k_0, \vec{y}) \oplus \mathsf{Eval}(k_1, \vec{y}) \neq 0^t$ otherwise.

On the other hand, if $\vec{y} \notin S_A$, we have $\mathsf{Eval}(k_0^{(i)}, \vec{y}) \neq \mathsf{Eval}(k_1^{(i)}, \vec{y})$ for each $i \in [\ell]$. Intuitively, for Alice to correctly compute the function $\mathsf{F}(\vec{y})$ in this case, she would need to guess all the ℓ choice bits sampled by Bob. Assuming H is a correlation robust hash function and setting $\ell = \Theta(\kappa)$ (where κ is the computational security parameter), the output $\mathsf{F}(\vec{y})$ is pseudorandom to Alice. Therefore, Alice only recognizes evaluations on values in the intersection.

Under this framework, the protocol inherently requires κ (e.g., 128) instantiations of the FSS scheme to achieve κ-bit security. This requirement appears *unavoidable* for security purposes; however, it incurs a *significant* overhead in both communication and computation costs. Thus, we ask the following question:

Can we design a new framework for sa-PSI that eliminates the need for repetition, thereby achieving concrete communication and computation efficiency?

New framework for sa-PSI. In this work, we propose a new framework for sa-PSI that eliminates the repetition bottleneck in prior constructions. At a high level, rather than having Alice generate both FSS key shares (k_0, k_1) and letting Bob learn one of them via OT, the two parties will instead jointly generate the FSS key shares such that each party only obtains one share. Specifically, we extend the set membership function (SMF) to be additionally parameterized by a string $\beta \in \{0,1\}^\kappa$:

$$\mathsf{SMF}_{S_A,\beta}(\vec{y}) = \begin{cases} 0^\kappa & \text{if } \vec{y} \in S \\ \beta & \text{otherwise} \end{cases}$$

The two parties perform a *distributed function secret sharing (dFSS)* protocol that jointly generates FSS key shares for $\mathsf{SMF}_{S_A,\beta}$, for which Alice inputs a structured set S_A and Bob inputs a random string $\beta \leftarrow\$ \{0,1\}^\kappa$. As a result, Alice obtains k_0 and Bob obtains k_1, with the guarantee that k_0 hides β from Alice and k_1 hides S_A from Bob. Correctness of the protocol ensures that:

$$\mathsf{Eval}(k_0, \vec{y}) \oplus \mathsf{Eval}(k_1, \vec{y}) = \begin{cases} 0^\kappa & \text{if } \vec{y} \in S \\ \beta & \text{otherwise} \end{cases}.$$

The rest of the protocol follows a similar approach as prior work. In particular, Bob now defines a function:

$$\mathsf{F}(\vec{y}) = \mathsf{H}\left(\mathsf{Eval}(k_1, \vec{y})\right),$$

and sends the set $Y := \{\mathsf{F}(\vec{y}) \mid \vec{y} \in S_B\}$ to Alice.

On Alice's side, we have the same guarantee that she can only compute this function for $\vec{y} \in S_A$; otherwise the output looks pseudorandom to her. This is because, for all points $\vec{y} \in S_A$, correctness of the dFSS protocol ensures that $\mathsf{Eval}(k_0, \vec{y}) = \mathsf{Eval}(k_1, \vec{y})$, hence Alice can compute $\mathsf{F}(\vec{y})$ as $\mathsf{H}\left(\mathsf{Eval}(k_0, \vec{y})\right)$. On the other hand, if $\vec{y} \notin S_A$, it holds that $\mathsf{Eval}(k_1, \vec{y}) = \mathsf{Eval}(k_0, \vec{y}) \oplus \beta$. Intuitively, for Alice to correctly compute the function $\mathsf{F}(\vec{y})$ in this case, she would need to guess the entire string β sampled by Bob. Thus, the output $\mathsf{F}(\vec{y})$ is pseudorandom to her, assuming H is a correlation robust hash function.

It is worth noting that our new framework achieves κ-bit security through the length of β, namely, the *output length* of the FSS, rather than relying on repetitions of FSS. In fact, it suffices for our framework to use a *relaxed* set membership function, where for all $\vec{y} \notin S$, the output of $\mathsf{SMF}_{S_A,\beta}(\vec{y})$ contains β as a substring, or at least κ bits of entropy. This relaxation will be useful for our concrete instantiation of the dFSS.

Incremental SMF and dFSS. Recall that Garimella et al. [GGM24] improve the aforementioned sa-PSI framework to achieve both computation and communication costs that scale with the description size of Alice's set S_A. Their main idea is to leverage a tool called *incremental FSS (iFSS)*, which enables evaluation not only on input strings but also on all their prefixes. They observe that a structured set S_A can be succinctly represented by a set of *critical prefixes*. By having Bob send iFSS evaluations on all prefixes of his inputs and having Alice evaluate on all her critical prefixes, Alice is able to identify matching prefixes and hence the intersection.

Our new framework is compatible with this technique, by extending the set membership function (SMF) to *incremental* SMF (iSMF), which can take any prefix as input (see Section 4 for details). By constructing dFSS for iSMF, our new framework naturally aligns with the approach of [GGM24].

Multiple balls with multiple points. Next, we consider the case where Alice's input is a union of multiple balls (or multiple structured sets), namely $S_A = \vec{S}_A^{(1)} \cup \ldots \cup \vec{S}_A^{(N_A)}$, and Bob's input S_B contains N_B points. Applying our framework to each ball in Alice's input gives a straightforward construction. However, as observed in prior works [GRS22, GRS23, GGM24], this naïve approach results in Bob's communication and computation costs scaling as $\mathcal{O}(N_A \cdot N_B)$, which is inefficient for large input sets.

Garimella et al. [GRS22] introduced a technique called *spatial hashing* to address this issue, reducing Bob's communication and computation to $\mathcal{O}(N_B)$. It was further improved in [GGM24] to handle overlapping balls (under certain conditions). At a high level, this technique divides the input space into contiguous grid cells, generates FSS keys for all active cells (i.e., cells that contain Alice's input balls), and then encodes these FSS keys into an oblivious key-value store (OKVS) [GPR+21].

However, this approach is incompatible with our framework. In more detail, under the previous sa-PSI framework [GRS22], Alice first assigns each ball $\vec{S}_A^{(i)}$ to a unique active cell $\vec{\sigma}^{(i)}$, and generates FSS shares $(k_0^{(i)}, k_1^{(i)}) \leftarrow \mathsf{Share}(1^\kappa, S_A^{(i)})$ for each $i \in [N_A]$. Then she encodes all the k_0's into OKVS_0 with key-value pairs $\{(\vec{\sigma}^{(i)}, k_0^{(i)})\}_{i\in[N_A]}$, and similarly encodes all the k_1's into OKVS_1 with key-value pairs $\{(\vec{\sigma}^{(i)}, k_1^{(i)})\}_{i\in[N_A]}$. This is followed by OT with Bob, allowing him to learn OKVS_b for a random choice bit $b \in \{0, 1\}$. The framework critically relies on Alice knowing both FSS shares in order to encode them into the OKVS. In contrast, our new sa-PSI framework applies a distributed FSS (dFSS) protocol where parties jointly generate FSS shares, Alice should learn only OKVS_0, and Bob only OKVS_1. This implies that we may need to incorporate the OKVS construction into the dFSS protocol, leading to prohibitive overhead.

Cuckoo hashing to the rescue. We resolve this issue by adopting the technique cuckoo hashing [Pag08]. Specifically, Alice constructs a cuckoo hash table for all her active cells, where each hash bin stores the corresponding ball (or empty). The two parties then perform dFSS for each bin of the hash table. On Bob's side, for each point $\vec{y} \in S_B$, he identifies all possible associated cells and, for each cell, all the possible hash bins. He then sends all the relevant FSS evaluations to Alice. If $\vec{y} \in S_A \cap S_B$, the correctness guarantees of cuckoo hashing and dFSS ensure that Alice will be able to identify the intersection.

There remains a subtle issue with this solution. When $\vec{y} \notin S_A \cap S_B$, and in particular, when $\vec{y}$ does not belong to any of Alice's active cells, it is still possible that $\vec{y}$ may be mapped to a hash bin that contains one of Alice's balls, allowing Alice to compute the evaluation of $\vec{y}$. This issue arises because different cells may be mapped to the same hash bin. To address this, we use an oblivious pseudorandom function (OPRF) to assign each cell a pseudorandom value, which is

then included in the input of the hash function H. This ensures that inactive cells are mapped to OPRF values unknown to Alice, making the final evaluations pseudorandom to her. We refer to Section Section 5.2 for more details.

Instantiation of our framework. We instantiate our sa-PSI framework with d-dimensional ℓ_∞ balls, for which the missing ingredient is a distributed FSS protocol tailored to the incremental set membership functions for such structured sets. In Section 6, we present a protocol based on pseudorandom generators and OT. Specifically, we construct a dFSS protocol for single-dimensional left-sided intervals, or equivalently, incremental distributed comparison functions (iDCF). Our protocol draws inspiration from Doerner-shelat [Ds17] and the iDCF construction by Garimella et al. [GGM24]. Compared to previous DCF constructions [BCG+22], we introduce an incremental property that may be of independent interest. In contrast to [GGM24], we provide a distributed key generation protocol for iDCF, making it compatible with our new framework.

To further improve concrete efficiency, we propose several optimizations. Since our iDCF is based on the GGM tree, it naturally fits the half-tree optimization [GYW+23], thereby reducing the number of PRG calls and improving concrete computational efficiency (Section 6.2). When constructing dFSS for multi-dimensional intervals, we introduce new techniques to reduce the exponential dependence on the dimension (Section 6.4). Finally, we notice the trade-offs between Alice's and Bob's workloads and propose methods to enable load-balancing between the two parties (Section 7.2).

3 Preliminaries

Notation. Let κ and λ denote the computational and statistical security parameters, respectively. For an integer $m \in \mathbb{N}$, let $[m] = \{1, \ldots, m\}$, for $a, b \in \mathbb{N}$ and $a < b$, $[a, b) = \{a, \ldots, b-1\}$. PPT stands for probabilistic polynomial time. By $\cong_\kappa$ and $\cong_\lambda$ we mean two distributions are computationally indistinguishable and statistically close, respectively. The notation $x \leftarrow\!\$\ \mathcal{D}$ denotes sampling from a probability distribution $\mathcal{D}$, with uniform sampling if $\mathcal{D}$ is a set.

Given a string $\alpha \in \{0,1\}^*$, we use $|\alpha|$ to denote its length, $\alpha_{[1:i]}$ to represent the prefix of length i of α, and α_i to the bit of α at i-th position, where $i \in [|\alpha|]$. For two strings $\alpha, \beta \in \{0,1\}^*$, we use $\alpha \| \beta$ to denote string concatenation. By $\alpha \prec \beta$, we mean that α is a prefix of β.

Hamming Correlation Robustness. Our protocol requires a hash function having the "correlation robustness" property.

Definition 1 (Correlation Robust Hash Function [IKNP03])**.** *Let $H : \{0,1\}^* \times \{0,1\}^\kappa \to \{0,1\}^v$ be a function and define the related function $F_s(t, x) = H(t; x \oplus s)$, where $s \in \{0,1\}^\kappa$. We say that H is correlation robust if F_s is computationally indistinguishable from a random function, against distinguishers that never query with repeated t-values. Intuitively, values of the form $H(t_i; x_i \oplus s)$ look jointly pseudorandom, even with known t_i, x_i values and an unknown but common s value.*

The specific property we use is defined below:

Definition 2. *Let $H : \{0,1\}^* \times (\{0,1\}^z)^\ell \to \{0,1\}^v$ be a function and define the related function $F_s(t, x, \Delta) = H(t; x \oplus (s \odot \Delta))$, where $s \in \{0,1\}^\ell$; x, Δ are vectors of length ℓ with components in $\{0,1\}^z$; and $\odot$ is component-wise multiplication (of a bit times a string). We say that H is **Hamming correlation robust** if F_s is computationally indistinguishable from a random function,*

against distinguishers that never query with repeated t-values and always query with Δ having at least κ nonzero components.

Intuitively, values of the form $H(t_i; x_i \oplus (s \odot \Delta_i))$ look jointly pseudorandom, even with known t_i, x_i, Δ_i values and an unknown but common s value, provided that each Δ_i has high Hamming weight.

This definition generalizes the one from [KKRT16] in that x and Δ are bit strings (vectors of bits) in [KKRT16], whereas in our protocol x and Δ can be vectors with components from $\{0,1\}^z$.

Cuckoo Hashing. Cuckoo hashing [Pag08] has been used in the literature on standard PSI [PSSZ15, KKRT16,PRTY20,BC23,RS21] to optimize both communication and computation of protocols. Below, we give a brief overview of this technique with a concrete choice of parameters to obtain a negligible failure probability.

Consider a set X having n items, k random hash functions $\mathsf{H}_1, \cdots, \mathsf{H}_k$, where $\mathsf{H}_i : \{0,1\}^* \to [m]$ and $m = \eta \cdot n$ for some expansion factor η. *Cuckoo hashing* with parameters (k, m, n) enables mapping a set of n items into a table T of size m using the hash functions $\{\mathsf{H}_i\}_{i\in[k]}$ such that each element $x \in X$ is put into one of the k hash bins $\mathsf{T}[\mathsf{H}_1(x)], \ldots \mathsf{T}[\mathsf{H}_k(x)]$, and that each bin in T contains at most one item. The failure probability of cuckoo hashing can be made negligible by carefully selecting parameters or by using a small stash to store the elements that cannot be placed in the hash table [PSWW18]. Following [CLR17, CMdG+21], we configure our scheme with the parameters $k = 3$ and omit the stash (see Section 7.3 for details of parameters choice).

Other cryptographic tools. We give definitions for other cryptographic tools including Oblivious Transfer (OT), pseudorandom generators (PRG), secure two-party computation (2PC), and OPRF in Appendix A.

4 Distributed Function Secret Sharing for Set Membership

4.1 Set Membership Functions

Notation. For a string $x \in \bigcup_{\ell=1}^{u}\{0,1\}^\ell$ and $\mathcal{U} = \{0,1\}^u$, we define the *prefix set* of x as $\mathsf{PreS}_x := \{s \mid x \prec s,\ s \in \mathcal{U}\}$, that is, the set of all strings in $\mathcal{U}$ with x as a prefix. For a vector $\vec{x} = (x_1, \ldots, x_d)$, where each $x_i \in \bigcup_{\ell=1}^{u}\{0,1\}^\ell$, we define $\mathsf{PreS}_{\vec{x}} := \{\vec{s} = (s_1, \ldots, s_d) \mid x_i \prec s_i,\ s_i \in \mathcal{U} \text{ for all } i \in [d]\}$. Note that $\mathsf{PreS}_{\vec{x}} = \mathsf{PreS}_{x_1} \times \cdots \times \mathsf{PreS}_{x_d}$.

Consider a family of d-dimensional sets $\mathcal{S} \subseteq \underbrace{2^{\mathcal{U}} \times \cdots \times 2^{\mathcal{U}}}_{d}$ where $\mathcal{U} = \{0,1\}^u$. As a concrete example, one can think of $\mathcal{S}$ as the family of all d-dimensional ℓ_∞-balls. For each d-dimensional set in the family $\mathcal{S}$, namely $\vec{S} = (S_1 \times \cdots \times S_d) \in \mathcal{S}$ (in the example, $\vec{S}$ is a d-dimensional ℓ_∞-ball) and an arbitrary string $\beta \in \{0,1\}^\kappa$, we define a **set membership function (SMF)** $\mathsf{SMF}_{\vec{S},\beta} : (\{0,1\}^u)^d \to \{0,1\}^\kappa$ that takes input $\vec{x} = (x_1, \ldots, x_d)$ where each $x_i \in \{0,1\}^u$, and outputs 0^κ if $\vec{x} \in \vec{S}$ and β otherwise. Formally,

$$\mathsf{SMF}_{\vec{S},\beta}(\vec{x}) = \begin{cases} 0^\kappa & \text{if } \vec{x} \in \vec{S} \\ \beta & \text{otherwise} \end{cases}.$$

Next, we define an ***incremental* set membership function (iSMF)** evaluated on all *prefixes* of $\vec{x}$. Specifically, for all $\vec{x} = (x_1, \ldots, x_d)$ where each $x_i \in \bigcup_{\ell=1}^{u}\{0,1\}^\ell$ can be of arbitrary length

(at most u bits), we define $\mathsf{iSMF}_{\vec{S},\beta}(\vec{x}) = 0^\kappa$ if the prefix $\vec{x}$ *spans* a set that is entirely contained in $\vec{S}$ (in our formal definition above, $\mathsf{PreS}_{\vec{x}} \subseteq \vec{S}$), and $\mathsf{iSMF}_{\vec{S},\beta}(\vec{x}) = \beta$ otherwise. Formally, $\mathsf{iSMF}_{\vec{S},\beta} : \left(\bigcup_{\ell=1}^{u}\{0,1\}^\ell\right)^d \to \{0,1\}^\kappa$ is defined as:

$$\mathsf{iSMF}_{\vec{S},\beta}(\vec{x}) = \begin{cases} 0^\kappa & \text{if } \mathsf{PreS}_{\vec{x}} \subseteq \vec{S} \\ \beta & \text{otherwise} \end{cases}.$$

Looking ahead, what we need for our new framework of structure-aware PSI is a ***relaxed* incremental set membership function (riSMF)** $\mathsf{riSMF}_{\vec{S},\beta} : \left(\bigcup_{\ell=1}^{u}\{0,1\}^\ell\right)^d \to \{0,1\}^\gamma$, which outputs a string of length $\gamma \geq \kappa$. For all inputs $\vec{x}$ such that $\mathsf{PreS}_{\vec{x}} \subseteq \vec{S}$, $\mathsf{riSMF}_{\vec{S},\beta}(\vec{x}) = 0^\gamma$; for all other $\vec{x}$, β appears as a *substring* in the output of $\mathsf{riSMF}_{\vec{S},\beta}(\vec{x})$, i.e., there exists $i \in [L]$ such that $\mathsf{riSMF}_{\vec{S},\beta}(\vec{x})_{[i\,:\,i+\kappa-1]} = \beta$.

4.2 riSMF for d-dimensional ℓ_∞-balls

In this work, we focus on riSMF for d-dimensional ℓ_∞-balls. As a special case, we first give the definition for single-dimensional left-sided intervals $S = \{x \mid x < \alpha,\ x \in \{0,1\}^u\}$, where $\alpha = \overline{\alpha_1\alpha_2\ldots\alpha_u} \in \{0,1\}^u$ (bit representation of α). The set S can be decomposed into a disjoint union of at most u prefix sets, that is,

$$S = \bigcup_{j\in[u]:\alpha_j=1} \mathsf{PreS}_{\overline{\alpha_1\ldots\alpha_{j-1}0}}.$$

We observe that $x < \alpha_{[1:|x|]}$ if and only if $\mathsf{PreS}_x \subseteq S$. Therefore, the incremental set membership function can be defined as an **incremental distributed comparison function** $\mathsf{iDCF}_{\alpha,\beta} : \bigcup_{\ell=1}^{u}\{0,1\}^\ell \to \{0,1\}^\kappa$. On input $x \in \bigcup_{\ell=1}^{u}\{0,1\}^\ell$,

$$\mathsf{iDCF}_{\alpha,\beta}(x) = \begin{cases} 0^\kappa & \text{if } x < \alpha_{[1:|x|]} \\ \beta & \text{otherwise} \end{cases}.$$

Next, consider a d-dimensional ℓ_∞-ball $\vec{S} = (S_1^{(s_1,t_1)} \times \cdots \times S_d^{(s_d,t_d)})$ represented by interval tuples $((s_1,t_1),\ldots,(s_d,t_d))$, where $s_i, t_i \in \{0,1\}^u$. A point $\vec{x} = (x_1,\ldots,x_d) \in (\{0,1\}^u)^d$ is in the set $\vec{S}$ iff $s_i < x_i < t_i$ for all $i \in [d]$. It suffices to consider left-sided intervals in a $2d$-dimensional space by mapping $\vec{x}$ to a $2d$-dimensional point $\vec{x}' = (x_1, \bar{x}_1, \ldots, x_d, \bar{x}_d)$, where $\vec{x} \in \vec{S}$ iff $x_i < t_i$ and $\bar{x}_i < \bar{s}_i$ for all $i \in [d]$.[2]

We define riSMF for $\vec{S}$ as a **relaxed incremental multi-dimensional comparison function** $\mathsf{riMCF}_{\vec{S},\beta} : \left(\bigcup_{\ell=1}^{u}\{0,1\}^\ell\right)^d \to \{0,1\}^{2d\cdot\kappa}$. On input $\vec{x} = (x_1,\ldots,x_d) \in \left(\bigcup_{\ell=1}^{u}\{0,1\}^\ell\right)^d$,

$$\mathsf{riMCF}_{\vec{S},\beta}(\vec{x}) = \beta_{1,0}^{(\vec{x})} \| \beta_{1,1}^{(\vec{x})} \| \ldots \| \beta_{d,0}^{(\vec{x})} \| \beta_{d,1}^{(\vec{x})},$$

where for each $i \in [d]$,

$$\beta_{i,0}^{(\vec{x})} = \begin{cases} 0^\kappa & \text{if } x_i < (t_i)_{[1:|x_i|]} \\ \beta & \text{otherwise} \end{cases}; \quad \beta_{i,1}^{(\vec{x})} = \begin{cases} 0^\kappa & \text{if } \bar{x}_i < (\bar{s}_i)_{[1:|x_i|]} \\ \beta & \text{otherwise} \end{cases}.$$

[2]For a string $z \in \{0,1\}^*$, we denote $\bar{z}$ as the bitwise complement of z, i.e., flipping all the bits of z. This ensures that for any two strings x, y of the same length, $x > y$ if and only if $\bar{x} < \bar{y}$.

Hence, if we evaluate $\mathsf{riMCF}_{\vec{S},\beta}$ on $\vec{x}$, the output is $0^{2d\cdot\kappa}$ if $\mathsf{PreS}_{\vec{x}} \subseteq \vec{S}$; otherwise, β appears as a substring in the output.

4.3 Distributed FSS for riSMF

We define the ideal functionality for generating secret shares of relaxed incremental set membership functions in Figure 1. The ideal functionality takes a structured input $\vec{S} \in \mathcal{S}$ from Alice and a string $\beta \in \{0,1\}^\kappa$ from Bob, and generates secret shares for $\mathsf{riSMF}_{\vec{S},\beta}(\vec{x})$ on every input $\vec{x}$. One can think of the process as a distributed FSS key generation for the function $\mathsf{riSMF}_{\vec{S},\beta}$.

Parameters:

- computational security parameter κ
- a family of structured set $\mathcal{S} \subseteq \underbrace{2^{\mathcal{U}} \times \cdots \times 2^{\mathcal{U}}}_{d}$ where $\mathcal{U} = \{0,1\}^u$
- a relaxed incremental set membership function defined for all $\vec{S} \in \mathcal{S}$ and $\beta \in \{0,1\}^\kappa$, $\mathsf{riSMF}_{\vec{S},\beta} : \left(\bigcup_{\ell=1}^{u} \{0,1\}^\ell\right)^d \to \{0,1\}^\gamma$

Functionality:

1. Receive (a succinct description of) structured input $\vec{S} \in \mathcal{S}$ from Alice.
2. Receive $\beta \in \{0,1\}^\kappa$ from Bob.
3. For each input $\vec{x} \in \left(\bigcup_{\ell=1}^{u} \{0,1\}^\ell\right)^d$:
   - Let $\mathbf{r}^{(\vec{x})} := \mathsf{riSMF}_{\vec{S},\beta}(\vec{x})$.
   - If both Alice and Bob are honest: sample secret shares $[\![\mathbf{r}^{(\vec{x})}]\!]_a, [\![\mathbf{r}^{(\vec{x})}]\!]_b \leftarrow\$ \{0,1\}^\gamma$ such that $[\![\mathbf{r}^{(\vec{x})}]\!]_a \oplus [\![\mathbf{r}^{(\vec{x})}]\!]_b = \mathbf{r}^{(\vec{x})}$;
   - If Alice (or Bob) is corrupt: receive $[\![\mathbf{r}^{(\vec{x})}]\!]_a$ from Alice (respectively, $[\![\mathbf{r}^{(\vec{x})}]\!]_b$ from Bob) and compute the other correlation share as $[\![\mathbf{r}^{(\vec{x})}]\!]_b = \mathbf{r}^{(\vec{x})} \oplus [\![\mathbf{r}^{(\vec{x})}]\!]_a$ (respectively, $[\![\mathbf{r}^{(\vec{x})}]\!]_b = \mathbf{r}^{(\vec{x})} \oplus [\![\mathbf{r}^{(\vec{x})}]\!]_a$).
4. Send $\left\{[\![\mathbf{r}^{(\vec{x})}]\!]_a\right\}_{\vec{x} \in \left(\bigcup_{\ell=1}^{u} \{0,1\}^\ell\right)^d}$ to Alice and $\left\{[\![\mathbf{r}^{(\vec{x})}]\!]_b\right\}_{\vec{x} \in \left(\bigcup_{\ell=1}^{u} \{0,1\}^\ell\right)^d}$ to Bob.

Figure 1: Ideal functionality $\mathcal{F}_{\mathsf{dFSS}}$ for generating secret shares of relaxed incremental set membership functions.

5 New sa-PSI Framework

In this section, we introduce our main structure-aware PSI protocol, where Alice holds a structured set and Bob holds a set of unstructured points. We present the protocol where Alice's input is a union of (overlapping) structures and Bob's input contains multiple points in Section 5.1 and the protocol with better efficiency using hashing scheme Section 5.2. Both constructions securely realize the functionality $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$ (Figure 2) in the presence of semi-honest adversaries with concrete proofs of security shown in Theorem 1 and Theorem 2, respectively.

5.1 Multiple (Overlapping) Balls with Multiple Points

We present an overview of our sa-PSI framework (see Figure 3) in a setting where Alice holds N_A structured sets $S_A = \bigcup_{i\in[N_A]} \vec{S}_A^{(i)}$ defined over a structured family $\mathcal{S}$, and Bob holds N_B points in the universe $(0,1^u)^d$. The protocol proceeds as follows:

- A distributed function secret sharing scheme $\mathcal{F}_{\mathsf{dFSS}}$ designed for the structured family $\mathcal{S}$ enables Alice and Bob to obtain shares of the function $\mathsf{riSMF}_{\vec{S}_A^{(i)},\beta}$ for each $i \in [N_A]$. These shares are the same for an input $\vec{x}$ if and only if $\vec{x} \in S_A$.
- A correlation-robust hash function H is used to hash Bob's FSS output evaluations. This ensures that Bob's input set remains hidden when he sends the set Y of these hashed values to Alice.
- An Intersection Search algorithm IntSearch (described in Algorithm 1) allows Alice to iteratively identify the intersection between her sets and Bob's input. Alice checks whether a corresponding FSS evaluation exists in Bob's hashed set Y for each critical prefix in her inputs. She then appends a 0 to the prefix and tests again; if the result is not in Y, she infers that the version of the prefix with a 1 must be present. This process repeats until the full matching input is recovered.

While both our construction and prior work [GGM24] have the IntSearch algorithm and the hash evaluations as performance bottlenecks, we introduce several key optimizations. In particular, we reduce the number of hashes sent by Bob to $\mathcal{O}(u)$, and the total number of IntSearch instances that Alice needs to perform to $\mathcal{O}(u)$. Both the computation and communication costs only grow with only $\mathcal{O}(u)$ instead of $\mathcal{O}(u^2)$ (see Section 6.4 for details).

Parameters: A family of sets $\mathcal{S} \subseteq \underbrace{2^{\mathcal{U}} \times \cdots \times 2^{\mathcal{U}}}_{d}$ where $\mathcal{U} = \{0,1\}^u$. Number of Alice's structured sets N_A and size of Bob's set N_B.

Functionality:

1. Receive input $S_A = \bigcup_{i\in N_A} \vec{S}_A^{(i)}$, where $\vec{S}_A^{(i)} \in \mathcal{S}$ (or a concise representation of S_A) from Alice.
2. Receive input $S_B \subseteq \mathcal{U}^d$ of size N_B from Bob.
3. [**output**] Send $S_A \cap S_B$ to Alice.

Figure 2: Ideal functionality $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$ for structure-aware PSI, where Alice learns the output.

Theorem 1. *For a set family $\mathcal{S} \subseteq \underbrace{2^{\mathcal{U}} \times \cdots \times 2^{\mathcal{U}}}_{d}$ where $\mathcal{U} = \{0,1\}^u$ and every set in $\mathcal{S}$ is a disjoint union of at most w prefix sets, given an ideal functionality for distributed function secret sharing $\mathcal{F}_{\mathsf{dFSS}}$ for $\mathsf{riSMF}_{\vec{S},\beta}$ (where $\vec{S} \in \mathcal{S}$ and $\beta \in \{0,1\}^\kappa$) and a Hamming correlation robust hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{\lambda+\log|N_A|+\log|N_B|+\log w+d\log u}$, the protocol in Figure 3 realizes $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$ (Figure 2) in the $\mathcal{F}_{\mathsf{dFSS}}$-hybrid model, in the presence of semi-honest adversaries.*

The details of the proof is shown in Appendix B.1.

5.2 Building Compatibility with Spatial Hashing Techniques

In our previous construction, the communication and computation costs of Bob depend on both set sizes of Alice and Bob, in this section, we propose a construction to optimize these costs by

Algorithm 1 Recursive Intersection Search

global variable: intersection set I
procedure $\textsc{IntSearch}(\vec{x}^*, \{[\![\mathbf{r}^{(\vec{x})}]\!]\}_{\vec{x}\in\mathsf{PreS}_{\vec{x}^*}}, u, \mathsf{aux}, Y)$
  Parse $\vec{x}^* = (x_1, \ldots, x_d)$
  Compute $h := \mathsf{H}\left(\vec{x}^* \| \mathsf{aux}; [\![\mathbf{r}^{(\vec{x}^*)}]\!]\right)$
  if $h \notin Y$ **then**
    return
  else if $|x_i| = u$ for all $i \in [d]$ **then**
    $I := I \cup \{\vec{x}\}$
    return
  Let $i \in [d]$ be the smallest index such that $|x_i| \leq u - 1$
  for $b = 0$ **to** 1 **do**
    $\vec{x}' := (x_1, \ldots, x_{i-1}, x_i \| b, x_{i+1}, \ldots, x_d)$
    $\textsc{IntSearch}(\vec{x}', \{[\![\mathbf{r}^{(\vec{x})}]\!]\}_{\vec{x}\in\mathsf{PreS}_{\vec{x}'}}, u, \mathsf{aux}, Y)$

dividing the universe into small cells under the assumption that each Alice's ball can be assigned to a unique mini-universe and cuckoo hashing enable Alice and Bob to distribute balls into the same mini-universe if there is a match.

Overview of Spatial Hashing. We follow the setup of [GGM24], where Alice holds a collection of d-dimensional ℓ_∞-balls of diameter δ, each represented as $\mathsf{Ball}_{\vec{x}} = [x_1, x_1+\delta) \times \cdots \times [x_d, x_d+\delta)$ for some "left-bottom corner" point $\vec{x} = (x_1, \ldots, x_d)$. The universe $(\{0,1\}^u)^d$ is partitioned into grid cells of side length δ. For each grid point $\vec{o} = (o_1, \ldots, o_d)$, where each o_i is a multiple of δ, we define a *mini-universe* (or *grid cell*) of diameter 2δ as $\mathsf{Univ}_{\vec{o}} := [o_1, o_1+2\delta) \times \cdots \times [o_d, o_d+2\delta)$. We assume that every ball $\mathsf{Ball}_{\vec{x}}$ fits entirely within some mini-universe $\mathsf{Univ}_{\vec{o}}$, and refer to such $\vec{o}$ as an *active origin* and the corresponding $\mathsf{Univ}_{\vec{o}}$ as an *active cell*. When constructing and evaluating our ibFSS in an active cell $\mathsf{Univ}_{\vec{o}}$, we *shift* all the points by the origin $\vec{o}$. Specifically, for $\vec{x} = (x_1, \ldots, x_d)$ and $\vec{o} = (o_1, \ldots, o_d)$, we consider $\vec{x}.\mathsf{Shift}(\vec{o}) := (x_1 - o_1, \ldots, x_d - o_d)$ in the mini-universe with diameter 2δ. When Bob computes the hash value, we include the origin $\vec{o}$ as an index to avoid collisions [GGM24].

Overview of our hashing technique. Alice then constructs a cuckoo hash table for all her active cells (cells) using three hash functions $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3$, where each hash bin stores the corresponding ball and its origin (or empty). Specifically, Alice creates a cuckoo hash table T using $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3$ on $\{\vec{o}_A^{(i)}\}_{i\in[N_A]}$, and put $(\vec{o}_A^{(i)}, \vec{S}_A^{(i)})$ into one of the hash bins $\mathsf{T}[\mathsf{H}_1(\vec{o}_A^{(i)})]$, $\mathsf{T}[\mathsf{H}_2(\vec{o}_A^{(i)})]$, or $\mathsf{T}[\mathsf{H}_3(\vec{o}_A^{(i)})]$. The two parties then perform dFSS for each bin of the hash table T.

On Bob's side, for each point $\vec{y} \in S_B$, he identifies all possible associated cells (there are 2^d of them), namely $\vec{y} \in \mathsf{Univ}_{\vec{o}}$ for origins $\vec{o}$; and each of these cells, all 3 possible hash bins. He then sends all the relevant FSS evaluations to Alice (for each Bob's ball, there are $3 \cdot (2\log(2\delta))^d$ hashed values). If $\vec{y} \in S_A \cap S_B$, Alice will be able to identify the intersection thanks to the correctness of cuckoo hashing and dFSS.

Protecting Bob's input balls. When $\vec{y} \notin S_A \cap S_B$, especially, when $\vec{y}$ does not belong to any of Alice's active cells, it is still possible that $\vec{y}$ may be mapped to a hash bin that contains one of Alice's balls, allowing Alice to compute the evaluation of $\vec{y}$. This issue comes from that different cells may be mapped to the same hash bin. To address this, we use an oblivious pseudorandom

Parameters:

- computational security parameter κ and statistical security parameter
- set family $\mathcal{S} \subseteq \underbrace{2^{\mathcal{U}} \times \cdots \times 2^{\mathcal{U}}}_{d}$ (where $\mathcal{U} = \{0,1\}^u$) with corresponding distributed function secret sharing $\mathcal{F}_{\mathsf{dFSS}}$ for $\mathsf{riSMF}_{\vec{S},\beta}$ (where $\vec{S} \in \mathcal{S}$ and $\beta \in \{0,1\}^\kappa$)
- Hamming correlation robust hash function H with output length $\lambda + \log|N_A| + \log|N_B| + \log w + d\log u$.

Inputs:

- Alice has N_A structured sets $S_A = \bigcup_{i \in N_A} \vec{S}_A^{(i)}$, where $\vec{S}_A^{(i)} \in \mathcal{S}$.
- Bob has an unstructured set $S_B \subseteq \mathcal{U}^d$ of size N_B.

Protocol:

1. Alice initializes the intersection as $I := \emptyset$.
2. Bob samples a random string $\beta \leftarrow\$ \{0,1\}^\kappa$.
3. For each $i \in [N_A]$, the parties do the following (in parallel for each set):
   (a) Alice and Bob invoke $\mathcal{F}_{\mathsf{dFSS}}$:
   - Alice inputs $\vec{S}_A^{(i)}$ and Bob inputs β.
   - Alice obtains $\left\{[\![\mathbf{r}_i^{(\vec{x})}]\!]_a\right\}_{\vec{x} \in \left(\bigcup_{\ell=1}^{u}\{0,1\}^\ell\right)^d}$ and Bob obtains $\left\{[\![\mathbf{r}_i^{(\vec{x})}]\!]_b\right\}_{\vec{x} \in \left(\bigcup_{\ell=1}^{u}\{0,1\}^\ell\right)^d}$.

   (b) Bob does the following:
   i. Initialize an empty set $Y := \emptyset$.
   ii. For each element $\vec{y} \in S_B$:
   - Write $\vec{y} = (y_1, \ldots, y_d)$, where $y_j \in \mathcal{U}$ for all $j \in [d]$.
   - Compute

$$Y' = \left\{ \mathsf{H}\left(\vec{y}' \| i; [\![\mathbf{r}_i^{(\vec{y}')}]\!]_b\right) \;\middle|\; \begin{array}{l} \ell_1 \in [u], \ldots, \ell_d \in [u] \\ \vec{y}' := ((y_1)_{[1:\ell_1]}, \ldots, (y_d)_{[1:\ell_d]}) \end{array} \right\}$$

   - Update $Y := Y \cup Y'$.

   iii. Pad Y with dummy random strings such that $|Y| = N_B \cdot u^d$ and send it to Alice.

   (c) Alice does the following:
   i. Write $S_A^{(i)} = \dot{\bigcup}_{j \in [w']} \mathsf{PreS}_{\vec{x}^j}$ as disjoint union of w' prefix sets (w is the upper bound for w').
   ii. For each $j \in [w']$, run $\textsc{IntSearch}\left(\vec{x}^j, \left\{[\![\mathbf{r}_i^{(\vec{x})}]\!]_a\right\}_{\vec{x} \in \mathsf{PreS}_{\vec{x}^j}}, u, i, Y\right)$, with I being a global variable.
4. **[output]** Alice outputs the intersection I.

Figure 3: Protocol realizing $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$ (Figure 2) in the $\mathcal{F}_{\mathsf{dFSS}}$-hybrid model.

function (OPRF) to assign each cell $\vec{o}$ a pseudorandom value as the output of PRF $\mathsf{PRF}(\vec{o})$, which is then included in the input of the hash function H. However, a further challenge arises because each cell is inserted into three possible bins (as in cuckoo hashing), therefore, to avoid collision between the cells that are assigned to different positions, we slightly modify the input of OPRF as $(t' \| \vec{o})$ where $t' = \mathsf{H}_\tau(\vec{o})$ is an index of $\vec{o}$ in the cuckoo hash table corresponding with the hash function H_τ for $\tau = \{1,2,3\}$.

Theorem 2. *For a set family $\mathcal{S} \subseteq \underbrace{2^{\mathcal{U}'} \times \cdots \times 2^{\mathcal{U}'}}_{d}$ where $\mathcal{U}' = 2\delta$ and every set in $\mathcal{S}$ is a disjoint union of at most w prefix sets, given an ideal functionality for distributed function secret sharing $\mathcal{F}_{\mathsf{dFSS}}$ for $\mathsf{riSMF}_{\vec{S},\beta}$ (where $\vec{S} \in \mathcal{S}$ and $\beta \in \{0,1\}^\kappa$), an ideal functionality for oblivious pseudoran-*

Parameters:

- computational security parameter κ and statistical security parameter λ
- set family $\mathcal{S} \subseteq \underbrace{2^{\mathcal{U}'} \times \cdots \times 2^{\mathcal{U}'}}_{d}$ (where $\mathcal{U}' = 2\delta$) with corresponding distributed function secret sharing $\mathcal{F}_{\mathsf{dFSS}}$ for $\mathsf{riSMF}_{\vec{S},\beta}$ (where $\vec{S} \in \mathcal{S}$ and $\beta \in \{0,1\}^\kappa$)
- OPRF ideal functionality $\mathcal{F}_{\mathsf{oprf}}$ (Figure 10)
- cuckoo hashing with hash functions $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3 : \mathcal{U}^d \to [m]$, where $m = \eta \cdot N_A$ is the size of cuckoo hash table.
- Hamming correlation robust hash function H with output length $\lambda + \log|N_A| + \log w + \log|N_B| + \log 3 + d(1 + \log\log(2\delta))$

Inputs:

- Alice has N_A structured sets $S_A = \bigcup_{i\in[N_A]} \vec{S}_A^{(i)}$, where $\vec{S}_A^{(i)} \in \mathcal{S}$.
- Bob has an unstructured set $S_B \subseteq \mathcal{U}^d$ of size N_B.

Notation:

- For $\vec{o} = (o_1, \ldots, o_d) \in \mathcal{U}^d$, $\mathsf{Univ}_{\vec{o}} := [o_1, o_1 + 2\delta) \times \ldots \times [o_d, o_d + 2\delta)$.
- For $\vec{x} = (x_1, \ldots, x_d) \in \mathcal{U}^d$, $\vec{o} := (o_1, \ldots, o_d) \in \mathcal{U}^d$, $\vec{x}.\mathsf{Shift}(\vec{o}) := (x_1 - o_1, \ldots, x_d - o_d)$.
- For $S \subseteq \mathcal{U}^d$ and $\vec{o} = (o_1, \ldots, o_d) \in \mathcal{U}^d$, $S.\mathsf{Shift}(\vec{o}) := \{\vec{x}.\mathsf{Shift}(\vec{o}) | \vec{x} \in S\}$.

Protocol:

1. Alice does the following:
   (a) Use spatial hashing to identify a distinct origin $\vec{o}_A^{(i)}$ such that $\vec{S}_A^{(i)} \subseteq \mathsf{Univ}_{\vec{o}_A^{(i)}}$ for each $i \in [N_A]$.
   (b) Create a cuckoo hash table T using $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3$ on $\{\vec{o}_A^{(i)}\}_{i\in[N_A]}$, and put $(\vec{o}_A^{(i)}, \vec{S}_A^{(i)})$ into one of the hash bins $\mathsf{T}[\mathsf{H}_1(\vec{o}_A^{(i)})]$, $\mathsf{T}[\mathsf{H}_2(\vec{o}_A^{(i)})]$, or $\mathsf{T}[\mathsf{H}_3(\vec{o}_A^{(i)})]$. Denote $t^{(i)}$ as the index of the hash bin for $(\vec{o}_A^{(i)}, \vec{S}_A^{(i)})$, namely $\mathsf{T}[t^{(i)}] = (\vec{o}_A^{(i)}, \vec{S}_A^{(i)})$.
2. Alice and Bob invoke $\mathcal{F}_{\mathsf{oprf}}$:
   - Alice inputs $(\mathsf{receiver}, \left\{(t^{(i)} \| \vec{o}_A^{(i)})\right\}_{i\in[N_A]})$ and Bob inputs (sender).
   - Alice receives $\{\mathsf{PRF}(t^{(i)} \| \vec{o}_A^{(i)})\}_{i\in[N_A]}$
3. Bob samples a random string $\beta \leftarrow\$ \{0,1\}^\kappa$.
4. For each $t \in [m]$, the parties do the following (in parallel for each hash bin):
   (a) Alice prepares a structured set $\vec{S}^{(t)}$ as follows:
      - If $\mathsf{T}[t]$ is non-empty, let $\mathsf{T}[t] = (\vec{o}_A^{(i)}, \vec{S}_A^{(i)})$ and compute $\vec{S}^{(t)} := \vec{S}_A^{(i)}.\mathsf{Shift}(\vec{o}_A^{(i)})$;
      - Otherwise, let $\vec{S}^{(t)} := \emptyset$.
   (b) Alice and Bob invoke $\mathcal{F}_{\mathsf{dFSS}}$:
      - Alice inputs $\vec{S}^{(t)}$ and Bob inputs β.
      - Alice obtains $\left\{[\![\mathbf{r}_t^{(\vec{x})}]\!]_a\right\}_{\vec{x}\in\left(\bigcup_{\ell=1}^{\log(2\delta)}\{0,1\}^\ell\right)^d}$ and Bob obtains $\left\{[\![\mathbf{r}_t^{(\vec{x})}]\!]_b\right\}_{\vec{x}\in\left(\bigcup_{\ell=1}^{\log(2\delta)}\{0,1\}^\ell\right)^d}$.
5. Bob initializes an empty set $Y := \emptyset$ and does the following:
   (a) For each element $\vec{y} \in S_B$, and for each origin $\vec{o}$ where $\vec{y} \in \mathsf{Univ}_{\vec{o}}$:
      i. Write $\vec{y}.\mathsf{Shift}(\vec{o}) = (y_1, \ldots, y_d)$, where $y_j \in [0, 2\delta)$ for all $j \in [d]$.
      ii. For each $\tau = \{1, 2, 3\}$, compute the position in cuckoo hash table $t' = \mathsf{H}_\tau(\vec{o})$.
         A. Send input $(\mathsf{sender}, t' \| \vec{o})$ to $\mathcal{F}_{\mathsf{oprf}}$ and receive $\mathsf{PRF}(t' \| \vec{o})$.
         B. Compute
$$Y' = \left\{ \mathsf{H}\left(\vec{y}' \| \vec{o} \| \mathsf{PRF}(t' \| \vec{o}); [\![\mathbf{r}_{t'}^{(\vec{y}')}]\!]_b\right) \middle| \begin{array}{l} \ell_1 \in [\log(2\delta)], \ldots, \ell_d \in [\log(2\delta)] \\ \vec{y}' := ((y_1)_{[1:\ell_1]}, \ldots, (y_d)_{[1:\ell_d]}) \end{array} \right\}$$
         C. Update $Y := Y \cup Y'$.
   (b) Pad Y with dummy random strings such that $|Y| = 3 \cdot N_B \cdot (2 \cdot \log(2\delta))^d$ and send it to Alice.
6. Alice initializes set $I := \emptyset$. For each $t \in [m]$ for which $\mathsf{T}[t]$ is non-empty, Alice does the following:
   (a) Let $\mathsf{T}[t] = (\vec{o}_A^{(i)}, \vec{S}_A^{(i)})$. Recall that $\vec{S}^{(t)} = \vec{S}_A^{(i)}.\mathsf{Shift}(\vec{o}_A^{(i)})$.
   (b) Write $\vec{S}^{(t)} = \dot{\bigcup}_{j\in[w']} \mathsf{PreS}_{\vec{x}^j}$ as disjoint union of w' prefix sets (w is the upper bound for w').
   (c) For each $j \in [w']$, run $\textsc{IntSearch}\left(\vec{x}^j, \left\{[\![\mathbf{r}_t^{(\vec{x})}]\!]_a\right\}_{\vec{x}\in\mathsf{PreS}_{\vec{x}^j}}, \log(2\delta), \vec{o}_A^{(i)} \| \mathsf{PRF}(t^{(i)} \| \vec{o}_A^{(i)}), Y\right)$, with I being a global variable.
7. **[output]** Alice outputs the intersection I.

Figure 4: Protocol realizing $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$ (Figure 2) using spatial hashing and cuckoo hashing techniques in the $(\mathcal{F}_{\mathsf{dFSS}}, \mathcal{F}_{\mathsf{oprf}})$-hybrid model.

dom functions $\mathcal{F}_{\mathsf{oprf}}$, a cuckoo hashing scheme with three hash functions $H_1, H_2, H_3 : \mathcal{U}^d \to [m]$ and no stash, and a Hamming correlation robust hash function

$$H : \{0,1\}^* \to \{0,1\}^{\lambda + \log|N_A| + \log w + \log|N_B| + \log 3 + d(1+\log\log(2\delta))},$$

the protocol in Figure 4 realizes $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$ (Figure 2) in the $(\mathcal{F}_{\mathsf{dFSS}}, \mathcal{F}_{\mathsf{oprf}})$-hybrid model, in the presence of semi-honest adversaries.

The details of proof can be found in Appendix B.2.

6 Distributed FSS for d-dimensional ℓ_∞-balls

In this section, we present constructions that realize $\mathcal{F}_{\mathsf{dFSS}}$ (Figure 1) for riSMF defined for d-dimensional ℓ_∞-balls (Section 4.2). We start with a construction for single-dimensional left-sided intervals in Section 6.1, and then extend it to d-dimensional ℓ_∞-balls in Section 6.3. We discuss additional optimizations in Section 6.2.

6.1 Construction of dFSS for iDCF

In this section, we present a construction for distributed FSS key generation for single-dimensional left-sided intervals of the form $\{x \mid x < \alpha,\ x \in \{0,1\}^u\}$. Recall from Section 4.2 the definition for an incremental distributed comparison function $\mathsf{iDCF}_{\alpha,\beta} : \bigcup_{\ell=1}^{u}\{0,1\}^\ell \to \{0,1\}^\kappa$. On input $x \in \bigcup_{\ell=1}^{u}\{0,1\}^\ell$, the function is defined as

$$\mathsf{iDCF}_{\alpha,\beta}(x) = \begin{cases} 0^\kappa & \text{if } x < \alpha_{[1:|x|]} \\ \beta & \text{otherwise} \end{cases}.$$

Construction Overview. As illustrated in Figure 5, our construction proceeds in two phases: key generation (steps 1–2) and full evaluation (steps 3–4). During key generation, Alice inputs $\alpha \in \{0,1\}^u$ and Bob inputs $\beta \in \{0,1\}^\kappa$, and they each obtain a succinct representation of additive shares of $\mathsf{iDCF}_{\alpha,\beta}(x)$ for all $x \in \bigcup_{\ell=1}^{u}\{0,1\}^\ell$. In the full evaluation phase, both parties can locally and deterministically compute their respective shares, $[\![\mathbf{r}^{(x)}]\!]_a$ and $[\![\mathbf{r}^{(x)}]\!]_b$, such that for all $x \in \bigcup_{\ell=1}^{u}\{0,1\}^\ell$,

$$[\![\mathbf{r}^{(x)}]\!]_a \oplus [\![\mathbf{r}^{(x)}]\!]_b = \begin{cases} 0^\kappa & \text{if } x < \alpha_{[1:|x|]} \\ \beta & \text{otherwise} \end{cases}.$$

The intuition behind the construction is that it emulates the "GGM tree" [GGM86] of depth u, where each string $x \in \bigcup_{\ell=1}^{u}\{0,1\}^\ell$ corresponds to a node at the $|x|$-th level of the tree. The tree is constructed by Bob as follows. He first randomly samples the root of the tree as $g_0^0 \leftarrow\$ \{0,1\}^\kappa$. Starting from the root, a pseudorandom generator (PRG) $G : \{0,1\}^\kappa \to \{0,1\}^{4\kappa}$ is applied on g_0^0 to generate $(g_0^1, g_1^1, c_0^1, c_1^1)$, each of κ bits, where $\{g_0^1, g_1^1\}$ are "base values" (used as PRG seeds for the next level) and $\{c_0^1, c_1^1\}$ are the associated "payloads." The tree is recursively expanded by applying G to each base value g_j^i at every level $i \in [u-1]$ and for every node $j \in [0, 2^i)$. Looking ahead, Bob locally defines his output share associated with each node x as $[\![\mathbf{r}^{(x)}]\!]_b = \bigoplus_{i=1}^{|x|} c^i_{x_{[1:i]}}$.

At a high level, the path from the root to the α-th leaf divides the tree into two partitions. Our construction ensures an *incremental sharing property*: if a node associated with x lies to the left

Parameters:

- computational security parameter κ
- pseudorandom generator $G : \{0,1\}^\kappa \to \{0,1\}^{4\kappa}$, with output split as $G(x) = G_0(x)\|G_1(x)\|C_0(x)\|C_1(x)$, each of κ bits
- ideal functionality for oblivious transfer $\mathcal{F}_{\mathsf{OT}}$ (Figure 9)

Inputs:

- Alice inputs $\alpha \in \{0,1\}^u$
- Bob inputs $\beta \in \{0,1\}^\kappa$

Protocol:

1. Bob does the following:
   (a) Randomly sample $g_0^0 \leftarrow\$ \{0,1\}^\kappa$.
   (b) Compute base values $\{g_j^i\}$ and payloads $\{c_j^i\}$ for all $i \in [u], j \in [0, 2^i)$ as follows:
      i. For each $i \in [u]$, compute the base values and payloads for the i-th level from the base values at the $(i-1)$-th level. Specifically, for all $j \in [0, 2^{i-1})$, compute
      $$(g_{j\|0}^i, g_{j\|1}^i, c_{j\|0}^i, c_{j\|1}^i) := G(g_j^{i-1}).$$
      ii. For each $i \in [u]$, compute the sums of the "left" and "right" base values, as well as sums of the "left" and "right" payloads:
      $$\mathbf{g}_0^i = \bigoplus_{j\in[0,2^{i-1})} g_{j\|0}^i, \quad \mathbf{g}_1^i = \bigoplus_{j\in[0,2^{i-1})} g_{j\|1}^i,$$
      $$\mathbf{c}_0^i = \bigoplus_{j\in[0,2^{i-1})} c_{j\|0}^i, \quad \mathbf{c}_1^i = \bigoplus_{j\in[0,2^{i-1})} c_{j\|1}^i.$$
2. Alice and Bob invoke u (parallel) instances of $\mathcal{F}_{\mathsf{OT}}$, where Bob is the sender and Alice is receiver.
   - For the first $\mathcal{F}_{\mathsf{OT}}$ instance, Alice inputs α_1, and Bob inputs the following messages:
   $$m_0^1 = (\mathbf{g}_1^1,\ \mathbf{c}_0^1 \oplus \beta,\ \mathbf{c}_1^1 \oplus \beta), \quad m_1^1 = (\mathbf{g}_0^1,\ \mathbf{c}_0^1,\ \mathbf{c}_1^1 \oplus \beta).$$
   - For the i-th instance of $\mathcal{F}_{\mathsf{OT}}$, where $i \in [2, u]$, Alice inputs α_i, and Bob inputs the following:
   $$m_0^i = (\mathbf{g}_1^i,\ \mathbf{c}_0^i,\ \mathbf{c}_1^i), \quad m_1^i = (\mathbf{g}_0^i,\ \mathbf{c}_0^i \oplus \beta,\ \mathbf{c}_1^i).$$

   For all $i \in [u]$, let $m^i = (\tilde{\mathbf{g}}^i, \tilde{\mathbf{c}}_0^i, \tilde{\mathbf{c}}_1^i)$ be the message that Alice receives from the i-th OT.
3. Alice recomputes the nodes in the "GGM tree" as follows (denote $\overline{\alpha_i} = \neg\alpha_i$):
   (a) For the first level, set
   $$\tilde{g}_{\overline{\alpha_1}}^1 = \tilde{\mathbf{g}}^1, \quad \tilde{c}_0^1 = \tilde{\mathbf{c}}_0^1, \quad \tilde{c}_1^1 = \tilde{\mathbf{c}}_1^1.$$
   (b) For each $i \in [2, u]$,
   - For each $j \in [0, 2^{i-1}),\ j \neq \alpha_{[1:i-1]}$, compute the base values and payloads as:
   $$(\tilde{g}_{j\|0}^i, \tilde{g}_{j\|1}^i, \tilde{c}_{j\|0}^i, \tilde{c}_{j\|1}^i) := G(\tilde{g}_j^i).$$
   - For $j = \alpha_{[1:i-1]}$ and each $b \in \{0,1\}$, compute the base value $\tilde{g}_{j\|\overline{\alpha_i}}^i$ and payloads $\tilde{c}_{j\|b}^i$ as:
   $$\tilde{g}_{j\|\overline{\alpha_i}}^i = \tilde{\mathbf{g}}^i \oplus \bigoplus_{j'\in[0,2^{i-1}),j'\neq j} \tilde{g}_{j'\|\overline{\alpha}_i}^i;$$
   $$\tilde{c}_{j\|b}^i = \tilde{\mathbf{c}}_b^i \oplus \bigoplus_{j'\in[0,2^{i-1}),j'\neq j} \tilde{c}_{j'\|b}^i.$$
4. **[output]** For each $x \in \bigcup_{\ell=1}^u \{0,1\}^\ell$:
   - Alice outputs $[\![\mathbf{r}^{(x)}]\!]_a = \bigoplus_{i=1}^{|x|} \tilde{c}_{x_{[1:i]}}^i$.
   - Bob outputs $[\![\mathbf{r}^{(x)}]\!]_b = \bigoplus_{i=1}^{|x|} c_{x_{[1:i]}}^i$.

Figure 5: Protocol Π_{iDCF} realizing $\mathcal{F}_{\mathsf{dFSS}}$ (Figure 1) for $\mathsf{iDCF}_{\alpha,\beta}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model.

of the path defined by α, then the parties receive shares of 0^κ; otherwise, they receive shares of β. To enable Alice to compute her share while preserving security, the parties perform a 1-out-of-2 oblivious transfer (OT) at each level $i \in [u]$, where Alice selects according to her input bit α_i, following the Doerner-shelat approach [Ds17]. From OT, Alice obtains a message $m^i = (\tilde{\mathbf{g}}^i, \tilde{\mathbf{c}}_0^i, \tilde{\mathbf{c}}_1^i)$ for each level $i \in [u]$, where $\tilde{\mathbf{g}}^i$ allows her to derive all the base values $\{g_j^i\}$ *except* those on the path corresponding to α, and $(\tilde{\mathbf{c}}_0^i, \tilde{\mathbf{c}}_1^i)$ allows her to derive all the payloads $\{c_j^i\}$. Specifically, Alice computes her share as $[\![\mathbf{r}^{(x)}]\!]_a = \bigoplus_{i=1}^{|x|} \tilde{c}^i_{x_{[1:i]}}$, which ensure that:

$$[\![\mathbf{r}^{(x)}]\!]_a \oplus [\![\mathbf{r}^{(x)}]\!]_b = \begin{cases} 0^\kappa & \text{if } x \text{ lies to the left of the path defined by } \alpha \text{ i.e., } x < \alpha_{[1:i]} \\ \beta & \text{otherwise} \end{cases}.$$

Theorem 3. *The protocol Π_{iDCF} presented in Figure 5 securely realizes $\mathcal{F}_{\mathsf{dFSS}}$ (Figure 1) for incremental distributed comparison functions $\mathsf{iDCF}_{\alpha,\beta}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model, in the presence of semi-honest adversaries.*

The proof is shown in Appendix B.3.

6.2 Half-tree Optimization

In this section, we present an optimization that reduces the computational overhead for our construction of Π_{iDCF}, based on a variant of GGM trees, called *correlated* GGM trees. Our optimized construction leverages the half-tree technique introduced by Guo at al. [GYW+23]. We briefly outline the optimized construction and security guarantees below.

The correlated GGM tree begins with a randomly sampled root $\Delta \leftarrow\$ \{0,1\}^\kappa$. In the first level, the left child X_0 is sampled randomly, and the right is defined as $X_0 \oplus \Delta$. For the remaining levels, each node $x \in \{0,1\}^\kappa$ is expanded by computing its left child as $\mathsf{H}(x)$ and the right child as $\mathsf{H}(x) \oplus x$. An example is illustrated in Figure 6. With this construction, the sums of all left nodes and all right nodes at each level have a constant offset Δ.

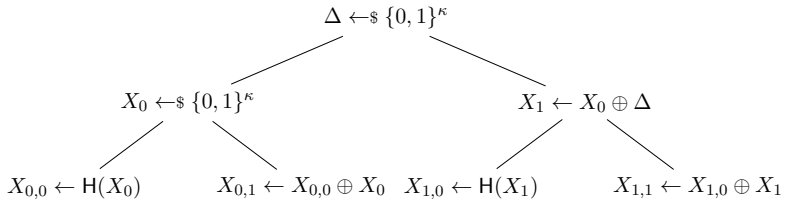

Figure 6: Example of a correlated GGM tree.

In our construction of Π_{iDCF} (Figure 5), we observe that for each level $i \in [u]$ of the GGM tree, Alice learns only either sum of the left or the right bases values $\{\mathbf{g}_0^i, \mathbf{g}_1^i\}$ as the output of OT instance corresponding to the input α_i. Consequently, the base values associated with prefixes of α remain hidden from Alice. This is a key property that enables a more efficient generation of the base values. Instead of calling two PRGs, G_0 and G_1, to compute both children's base values from a parent node, we use the correlated tree construction to reduce the number of PRG calls by half. The correctness of the protocol is unaffected by this optimization. The intuition behind its security is that the offset Δ always remains hidden from Alice's viewpoint throughout the protocol. As shown in [GYW+23], pseudorandomness is preserved assuming the hash function H satisfies

the circular correlation robustness (CCR) property. Therefore, the security analysis of Theorem 3 remains unchanged.

6.3 Construction of dFSS for riMCF

In this section, we present a construction for distributed FSS key generation for d-dimensional ℓ_∞-balls of the form $\vec{S} = (S_1^{(s_1,t_1)} \times \cdots \times S_d^{(s_d,t_d)})$ represented by interval tuples $((s_1,t_1),\ldots,(s_d,t_d))$, where $s_i, t_i \in \{0,1\}^u$. A point $\vec{x} = (x_1,\ldots,x_d) \in (\{0,1\}^u)^d$ is in the set $\vec{S}$ iff $s_i < x_i < t_i$ for all $i \in [d]$. Recall from Section 4.2 the definition for a relaxed incremental multi-dimensional comparison function $\mathsf{riMCF}_{\vec{S},\beta} : \left(\bigcup_{\ell=1}^{u}\{0,1\}^\ell\right)^d \to \{0,1\}^{2d\cdot\kappa}$. On input $\vec{x} = (x_1,\ldots,x_d) \in \left(\bigcup_{\ell=1}^{u}\{0,1\}^\ell\right)^d$,

$$\mathsf{riMCF}_{\vec{S},\beta}(\vec{x}) = \beta_{1,0}^{(\vec{x})} \| \beta_{1,1}^{(\vec{x})} \| \ldots \| \beta_{d,0}^{(\vec{x})} \| \beta_{d,1}^{(\vec{x})},$$

where for each $i \in [d]$,

$$\beta_{i,0}^{(\vec{x})} = \begin{cases} 0^\kappa & \text{if } x_i < (t_i)_{[1:|x_i|]} \\ \beta & \text{otherwise} \end{cases}; \quad \beta_{i,1}^{(\vec{x})} = \begin{cases} 0^\kappa & \text{if } \bar{x}_i < (\bar{s}_i)_{[1:|x_i|]} \\ \beta & \text{otherwise} \end{cases}.$$

Parameters:
- computational security parameter κ
- ideal functionality $\mathcal{F}_{\mathsf{iDCF}}$ for generating secret shares of $\mathsf{iDCF}_{\alpha,\beta}$

Inputs:
- Alice inputs $\vec{S} = (S_1^{(s_1,t_1)} \times \cdots \times S_d^{(s_d,t_d)})$
- Bob inputs $\beta \in \{0,1\}^\kappa$

Protocol:
1. For each dimension $i \in [d]$, parties do the following (in parallel for all invocations of $\mathcal{F}_{\mathsf{iDCF}}$):
   (a) Invoke $\mathcal{F}_{\mathsf{iDCF}}$:
   - Alice inputs $\alpha = t_i$ and Bob inputs β
   - Alice obtains $\left\{[\![\mathbf{r}_{i,0}^{(x)}]\!]_a\right\}_{x\in\bigcup_{\ell=1}^{u}\{0,1\}^\ell}$ and Bob obtains $\left\{[\![\mathbf{r}_{i,0}^{(x)}]\!]_b\right\}_{x\in\bigcup_{\ell=1}^{u}\{0,1\}^\ell}$

   (b) Invoke $\mathcal{F}_{\mathsf{iDCF}}$:
   - Alice inputs $\alpha = \bar{s}_i$ and Bob inputs β
   - Alice obtains $\left\{[\![\mathbf{r}_{i,1}^{(x)}]\!]_a\right\}_{x\in\bigcup_{\ell=1}^{u}\{0,1\}^\ell}$ and Bob obtains $\left\{[\![\mathbf{r}_{i,1}^{(x)}]\!]_b\right\}_{x\in\bigcup_{\ell=1}^{u}\{0,1\}^\ell}$
2. **[output]** For each $\vec{x} = (x_1,\ldots,x_d) \in \left(\bigcup_{\ell=1}^{u}\{0,1\}^\ell\right)^d$:
   - Alice outputs $[\![\mathbf{r}^{(\vec{x})}]\!]_a = [\![\mathbf{r}_{1,0}^{(x_1)}]\!]_a \| [\![\mathbf{r}_{1,1}^{(\bar{x}_1)}]\!]_a \| \ldots \| [\![\mathbf{r}_{d,0}^{(x_d)}]\!]_a \| [\![\mathbf{r}_{d,1}^{(\bar{x}_d)}]\!]_a$.
   - Bob outputs $[\![\mathbf{r}^{(\vec{x})}]\!]_b = [\![\mathbf{r}_{1,0}^{(x_1)}]\!]_b \| [\![\mathbf{r}_{1,1}^{(\bar{x}_1)}]\!]_b \| \ldots \| [\![\mathbf{r}_{d,0}^{(x_d)}]\!]_b \| [\![\mathbf{r}_{d,1}^{(\bar{x}_d)}]\!]_b$.

Figure 7: Protocol Π_{riMCF} that realizes $\mathcal{F}_{\mathsf{dFSS}}$ (Figure 1) for $\mathsf{riMCF}_{\vec{S},\beta}$ in the $\mathcal{F}_{\mathsf{iDCF}}$-hybrid model.

Construction Overview. Notice that the output of $\mathsf{riMCF}_{\vec{S},\beta}(\vec{x})$ is a concatenation of $2d$ single-dimensional left-sided interval functions. Therefore, given a distributed FSS for single-dimensional left-sided intervals, i.e., $\mathcal{F}_{\mathsf{iDCF}}$, we can simply apply it to each dimension and concatenate all the resulting FSS shares, as shown in Figure 7.

Theorem 4. *The protocol Π_{riMCF} presented in Figure 7 securely realizes $\mathcal{F}_{\mathsf{dFSS}}$ (Figure 1) for relaxed incremental multi-dimensional comparison functions $\mathsf{riMCF}_{\vec{S},\beta}$ in the $\mathcal{F}_{\mathsf{iDCF}}$-hybrid model, in the presence of semi-honest adversaries.*

6.4 Reducing Exponential Dependence on $2d$

Following the approach of [GGM24], when instantiating our new sa-PSI framework with the dFSS construction for d-dimensional ℓ-infinity balls in Section 6.3, both the computation and communication costs grow exponentially with $2d$. In this section, we present an optimization that reduces the exponential dependence to d, which significantly improves concrete efficiency especially in low dimensions.

We start by reviewing the ideas of [GGM24] for single-dimensional two-sided intervals. Alice holds an interval (s,t) where $s,t \in \{0,1\}^u$, and Bob holds a single point $y \in \{0,1\}^u$. Following our protocol in Figure 7, both parties need to invoke two $\mathcal{F}_{\mathsf{dFSS}}$ instances, where Alice's inputs are t and $\bar{s}$, respectively, producing shares $(\llbracket \mathbf{r}_0^{(x)} \rrbracket_a, \llbracket \mathbf{r}_0^{(x)} \rrbracket_b)$ and $(\llbracket \mathbf{r}_1^{(x)} \rrbracket_a, \llbracket \mathbf{r}_1^{(x)} \rrbracket_b)$ for all $x \in \bigcup_{\ell=1}^{u} \{0,1\}^\ell$.

Recall from Section 4.2 that for each one-sided interval $[0,\alpha)$, where $\alpha = \overline{\alpha_1 \alpha_2 \dots \alpha_u} \in \{0,1\}^u$ (bit representation of α), the structured set $S := \{x \mid x < \alpha,\ x \in \{0,1\}^u\}$ can be decomposed into a disjoint union of at most u prefix sets, namely $S = \bigcup_{j \in [u]:\alpha_j = 1} \mathsf{PreS}_{\overline{\alpha_1 \dots \alpha_{j-1} 0}}$. We define a set of critical prefixes $\mathsf{CrtPre}(\alpha)$ as

$$\mathsf{CrtPre}(\alpha) = \left\{ \overline{\alpha_1 \dots \alpha_{j-1} 0} \mid j \in [u] : \alpha_j = 1 \right\}.$$

On Alice's side, she considers her critical prefixes for (s,t) as all pairs of critical prefixes for $[0,t)$ and $[0,\bar{s})$, namely,

$$\mathsf{CrtPre}_{\mathsf{old}}(s,t) := \mathsf{CrtPre}(t) \times \mathsf{CrtPre}(\bar{s}).$$

This is according to the following observation:

Observation 1 ([GGM24]). *If $y \in (s,t)$, there exists a pair of critical prefixes $(t',s') \in \mathsf{CrtPre}_{\mathsf{old}}(s,t)$ such that $t' \prec y$ and $s' \prec \bar{y}$. Moreover, for any (y_0,y_1) where $t' \prec y_0 \prec y$ and $s' \prec y_1 \prec \bar{y}$, we have*

$$\llbracket \mathbf{r}_0^{(y_0)} \rrbracket_a \| \llbracket \mathbf{r}_1^{(y_1)} \rrbracket_a = \llbracket \mathbf{r}_0^{(y_0)} \rrbracket_b \| \llbracket \mathbf{r}_1^{(y_1)} \rrbracket_b.$$

If Bob sends Alice hash values for all pairs of prefixes of y and $\bar{y}$:

$$Y_{\mathsf{old}} = \left\{ \mathsf{H}\left(y_0 \| y_1 \| \llbracket \mathbf{r}_0^{(y_0)} \rrbracket_b \| \llbracket \mathbf{r}_1^{(y_1)} \rrbracket_b \right) \middle| \begin{array}{l} y_0 \prec y \\ y_1 \prec \bar{y} \end{array} \right\},$$

then Alice can perform IntSearch on input (s',t',Y_{old}) for each $(s',t') \in \mathsf{CrtPre}_{\mathsf{old}}(s,t)$ to identify the intersection.

In this approach, Bob needs to send $\mathcal{O}(u^2)$ hash values, and Alice needs to consider $\mathcal{O}(u^2)$ pairs of critical prefixes. Both computation and communication costs grow with $\mathcal{O}(u^2)$. In a d-dimensional space, this grows to $\mathcal{O}(u^{2d})$. Next, we discuss an optimization that reduces the exponential dependence from $2d$ to d.

Redefining critical prefixes. Rather than considering the *product* of all critical prefixes for $[0,t)$ and $[0,\bar{s})$, we redefine Alice's critical prefixes for (s,t) as the *union* of the two sets, namely,

$$\mathsf{CrtPre}_{\mathsf{new}}(s,t) := \mathsf{CrtPre}(t) \cup \mathsf{CrtPre}(\bar{s}).$$

This is backed by the following novel observation:

Observation 2. *If $y \in (s,t)$, there exists $z \in \mathsf{CrtPre}_{\mathsf{new}}(s,t)$, such that at least one of the following conditions holds:*
(1) $z \prec y$, and for any $z \prec y' \prec y$, we have

$$[\![\mathbf{r}_0^{(y')}]\!]_a \| [\![\mathbf{r}_1^{(\bar{y}')}]\!]_a = [\![\mathbf{r}_0^{(y')}]\!]_b \| [\![\mathbf{r}_1^{(\bar{y}')}]\!]_b.$$

(2) $z \prec \bar{y}$, and for any $z \prec y' \prec \bar{y}$, we have

$$[\![\mathbf{r}_0^{(\bar{y}')}]\!]_a \| [\![\mathbf{r}_1^{(y')}]\!]_a = [\![\mathbf{r}_0^{(\bar{y}')}]\!]_b \| [\![\mathbf{r}_1^{(y')}]\!]_b.$$

Thus, Bob only needs to send Alice hash values for all prefixes of y and $\bar{y}$:

$$\begin{aligned} Y_{\mathsf{new}} = &\left\{ \mathsf{H}\left(y' \| \bar{y}' \| [\![\mathbf{r}_0^{(y')}]\!]_b \| [\![\mathbf{r}_1^{(\bar{y}')}]\!]_b \right) \middle| y' \prec y \right\} \\ &\cup \left\{ \mathsf{H}\left(\bar{y}' \| y' \| [\![\mathbf{r}_0^{(\bar{y}')}]\!]_b \| [\![\mathbf{r}_1^{(y')}]\!]_b \right) \middle| y' \prec \bar{y} \right\}. \end{aligned}$$

Then Alice can perform IntSearch on input (z, Y_{new}) for each $z \in \mathsf{CrtPre}_{\mathsf{new}}(s,t)$.

This approach reduces the number of hashes sent by Bob to $\mathcal{O}(u)$, and the total number of IntSearch instances that Alice needs to perform to $\mathcal{O}(u)$. Both the computation and communication costs only grow with $\mathcal{O}(u)$. This method naturally extends to d dimension, where the costs scale as $\mathcal{O}(u^d)$ rather than $\mathcal{O}(u^{2d})$.

7 Performance

7.1 Theoretical Comparison

In this section, we give an asymptotic comparison of the communication and computation costs between our sa-PSI construction and prior works.

sa-PSI from FSS. We first compare our new framework with the FSS-based semi-honest sa-PSI protocols [GRS22,GGM24]. Table 1 presents a comparison in the plain setting (Section 5.1), where Alice holds N_A number of ℓ_∞ balls in the d-dimensional universe $(\{0,1\}^u)^d$ and Bob holds N_B points in the same universe. Table 2 shows a comparison in the spatial hashing setting (Section 5.2), where Alice holds N_A number of ℓ_∞-balls of diameter δ in the d-dimensional universe $(\{0,1\}^u)^d$ and Bob holds N_B points in the universe. Spatial hashing techniques are applied to effectively reduce the costs.

The communication cost can be broken down into two parts: (1) OT for distributing FSS keys, and (2) hash values sent from Bob to Alice. As shown in both tables, our new framework achieves a κ-fold improvement in the communication cost of distributing FSS keys via OT.

The computational cost consists of FSS key generation and evaluation, Bob's computation of hashes, and Alice's search for intersection. Notably, all the computation only relies on efficient symmetric-key operations such as PRGs (which can be instantiated with AES) and hash functions. As shown in both tables, our construction again achieves a κ-fold improvement in both Bob's hash computation and Alice's intersection search. For FSS key generation, our construction introduces a trade-off. On the one hand, we require the number of PRG calls to grow linearly with the (mini-)universe size, i.e., 2^u or δ, whereas prior works only grow with u or $\log \delta$. On the other

Comp. & Comm. Costs			[GRS22]	[GGM24]	Ours
Comp.	FSS (PRG calls)	Alice	$\mathcal{O}(\kappa \cdot N_A \cdot u^d)$	$\mathcal{O}(\kappa \cdot N_A \cdot u \cdot d)$	$\mathcal{O}(N_A \cdot 2^u \cdot d)$
		Bob	$\mathcal{O}(\kappa \cdot N_A \cdot N_B \cdot u^d)$	$\mathcal{O}(\kappa \cdot N_A \cdot N_B \cdot u \cdot d)$	
	Alice's Intersection		$\mathcal{O}(\kappa \cdot \lvert S_A \rvert \cdot N_A \cdot u^d)$	$\mathcal{O}(N_A \cdot u^d + \kappa \cdot N_B \cdot u \cdot d)$	$\mathcal{O}(N_A \cdot u^d + N_B \cdot u \cdot d)$
	Bob's Hashes		$\mathcal{O}(\kappa \cdot N_A \cdot N_B \cdot u^d)$	$\mathcal{O}(N_A \cdot N_B \cdot (u^d + \kappa \cdot u \cdot d))$	$\mathcal{O}(N_A \cdot N_B \cdot (u^d + u \cdot d))$
Comm. (bits)	OT		$\mathcal{O}(\kappa^2 \cdot N_A \cdot d \cdot u^d)$	$\mathcal{O}(\kappa^2 \cdot N_A \cdot u \cdot d)$	$\mathcal{O}(\kappa \cdot N_A \cdot u \cdot d)$
	Bob's Hashes		$\mathcal{O}(N_A \cdot N_B \cdot h_{\text{out}})$	$\mathcal{O}(N_A \cdot N_B \cdot u^d \cdot h_{\text{out}})$	

Table 1: Computation and communication costs comparing our protocol with prior sa-PSI constructions in the setting where Alice holds N_A number of ℓ_∞ balls in the d-dimensional universe $(\{0,1\}^u)^d$ and Bob holds N_B points in the universe. $h_{\text{out}} = \mathcal{O}(\lambda)$ is the output length of the final hash function. $|S_A|$ is upper bounded by $2^{u \cdot d}$.

Comp. & Comm. Costs			[GRS22]	[GGM24]	Ours
Comp.	FSS (PRG calls)	Alice	$\mathcal{O}(\kappa \cdot N_A \cdot (4 \log \delta)^d)$	$\mathcal{O}(\kappa \cdot N_A \cdot \log \delta \cdot d)$	$\mathcal{O}(N_A \cdot \delta \cdot d)$
		Bob	$\mathcal{O}(\kappa \cdot N_B \cdot (4 \log \delta)^d)$	$\mathcal{O}(\kappa \cdot N_B \cdot \log \delta \cdot d)$	
	Alice's Intersection		$\mathcal{O}(\kappa \cdot \lvert S_A \rvert \cdot N_A \cdot (2 \log \delta)^d)$	$\mathcal{O}(N_A \cdot (\log \delta)^d + \kappa \cdot N_B \cdot \log \delta \cdot d)$	$\mathcal{O}(N_A \cdot (\log \delta)^d + N_B \cdot \log \delta \cdot d)$
	Bob's Hashes		$\mathcal{O}(\kappa \cdot N_B \cdot (2 \log \delta)^d)$	$\mathcal{O}(N_B \cdot 2^d \cdot ((\log \delta)^d + \kappa \cdot \log \delta \cdot d))$	$\mathcal{O}(N_B \cdot 2^d \cdot ((\log \delta)^d + \log \delta \cdot d))$
Comm. (bits)	OT		$\mathcal{O}(\kappa^2 \cdot N_A \cdot d \cdot (4 \log \delta)^d)$	$\mathcal{O}(\kappa^2 \cdot N_A \cdot \log \delta \cdot d)$	$\mathcal{O}(\kappa \cdot N_A \cdot \log \delta \cdot d)$
	Bob's Hashes		$\mathcal{O}(N_B \cdot h_{\text{out}})$	$\mathcal{O}(N_B \cdot (2 \log \delta)^d \cdot h_{\text{out}})$	

Table 2: Computation and communication costs comparing our protocol with prior sa-PSI constructions in the spatial hashing setting where Alice holds N_A number of ℓ_∞ balls of diameter δ in the d-dimensional universe $(\{0,1\}^u)^d$ and Bob holds N_B points in the universe. $h_{\text{out}} = \mathcal{O}(\lambda)$ is the output length of the final hash function. $|S_A|$ is upper bounded by δ^d.

hand, because we employ distributed FSS key generation, we eliminate the κ factor required when Alice generates both shares on her own. Moreover, the number of PRG calls on Bob's side is independent of his set size N_B. Since PRG calls can be efficiently instantiated using fixed-key AES, which benefits from hardware acceleration, and the diameter δ is typically small in practice, this trade-off is quite reasonable and practical.

Fuzzy PSI for ℓ_∞ distance. We provide a comparison of our construction with prior fuzzy PSI protocols for ℓ_∞ distance [vBP24, GQL+24, GGM24, GRS22, RRX24] in Table 3.

Input Assumptions. To optimize both communication and computation costs, all existing constructions rely on certain assumptions about the distribution of input sets. The public-key-based works [vBP24, GQL+24] rely on the disjoint projection assumption (disj. proj.), which states that for any pair of balls from Alice and Bob, there exists at least one dimension in which the components differ by distance more than 2δ. The work of [vBP24] also supports an alternative distance assumption (e.g., a minimum distance between balls). The work of [GQL+24] further enforces the disjoint projection condition on both Alice's and Bob's inputs. The recent work of [RRX24] requires a disjoint hashing assumption (disj. hash), which requires that if two balls are not "close," the sets of bins they are hashed into do not overlap.

In contrast, [GRS22, GGM24] and our work do not require any assumptions on Bob's input.

Protocol	Input Assumptions		Comp. Assumptions	Communication	Computation	
	Alice	Bob			Alice	Bob
[vBP24]	$\ell_{\min} > 2\delta$	✓	DDH	$\mathcal{O}\left(\delta d N_A + 2^d N_B\right)$	$\mathcal{O}\left(\delta d N_A + 2^d N_B\right)$	$\mathcal{O}\left(2^d d N_B\right)$
	disj. proj.			$\mathcal{O}\left((\delta d)^2 N_A + N_B\right)$	$\mathcal{O}\left(d^2 N_A + N_B\right)$	$\mathcal{O}\left(d^2 N_B\right)$
[GQL+24]	disj. proj.		AHE, DDH	$\mathcal{O}\left(\delta d(N_A + N_B)\right)$	$\mathcal{O}\left(\delta d N_A + N_B\right)$	$\mathcal{O}\left(N_A + \delta d N_B\right)$
[RRX24]	disj. hash		OT	$\mathcal{O}\left(\log \delta d(2^s N_A + 2^{d-s} N_B)\right)$	$\Omega(\log \delta d 2^s N_A)$	$\Omega(\log \delta d 2^{d-s} N_B)$
[GRS22]	$\ell_{\min} > 2\delta$	✓	OT	$\mathcal{O}\left((4 \log \delta)^d d N_A + N_B\right)$	$\mathcal{O}\left((4 \log \delta)^d N_A\right)$	$\mathcal{O}\left((2 \log \delta)^d N_B\right)$
[GGM24] Ours	unique mini-univ.	✓	OT	$\mathcal{O}\left(\log \delta d N_A + (2 \log \delta)^d N_B\right)$	$\mathcal{O}((\log \delta)^d N_A + \log \delta d N_B)$	$\mathcal{O}((2 \log \delta)^d N_B)$

Table 3: Comparison of fuzzy PSI protocols, where Alice holds N_A number of ℓ_∞ balls of diameter δ in the d-dimensional universe $(\{0,1\}^u)^d$ and Bob holds N_B points in the universe. The assumption highlighted in red only holds for significantly large d. ✓ means there is no assumption required.

The construction of [GRS22] requires a minimum distance condition between balls, i.e., $\ell_{\min} > 2\delta$. Meanwhile, both our construction and [GGM24] only require that each ball maps uniquely to a distinct mini-universe, i.e., the active mini-universe of the balls are non-colliding, but the balls themselves may overlap.

Computational Assumptions. [vBP24, GQL+24] require public-key assumptions such as AHE and DDH, whereas [GRS22, GGM24] and our work are based on symmetric-key primitives and OT. The recent work of [RRX24] also relies solely on symmetric-key tools, using garbled circuits.

Computation and Communication Costs. Compared to the public-key-based approaches [vBP24, GQL+24], their computation and communication costs grow quadratically or linearly in the dimension d, while ours grow exponentially. On the other hand, their costs grow linearly in the diameter δ, whereas ours only grow polylogarithmically. Additionally, they require computationally expensive public-key operations whereas we rely on efficient symmetric-key primitives. Hence we expect our approach to outperform theirs in low dimensions and large diameters, which will be demonstrated in our experimental results.

7.2 Load Balancing Between Alice and Bob

Notice in Table 1 that [GGM24] improves upon [GRS22] by reducing Alice's computational cost, at the expense of requiring Bob to send more hash values as "hints." This introduces a tradeoff between Bob's workload (in both computation and communication) and Alice's computational cost. In our implementation, we propose a new method to achieve load balancing between Alice's and Bob's workloads by modifying IntSearch as follows:

- Bob sends only the hashes for prefixes with lengths in $\mathcal{L}$, that is, $\vec{y}' = (y'_1, \ldots, y'_d)$ where $y'_i \prec y_i$ and $|y_i| \in \mathcal{L}$ for each $i \in [d]$. This reduces the number of hashes sent by Bob from $(\log(2\delta))^d$ to $|\mathcal{L}|^d$ (see Table 6 for the parameters used in our experiments).
- Alice runs a modified IntSearch algorithm to expand the search for prefixes not in $\mathcal{L}$. For an input $\vec{x}^* = (x_1, \ldots, x_d)$, let $(\ell_1, \ldots, \ell_d)$ be the tuple of lengths such that each ℓ_i is the smallest length satisfying $\ell_i \geq |x_i|$ and $\ell_i \in \mathcal{L}$. Alice then runs IntSearch recursively on each $(x'_1, \ldots, x'_d)$ where $x_i \prec x'_i$ and $|x'_i| = \ell_i$ for each $i \in [d]$. This leads to 2^t recursive calls to IntSearch, where $t = \sum_{i=1}^{d} (\ell_i - |x_i|)$.

7.3 Implementation Details

We implement our protocols in Rust.[3] We choose the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$.

Cryptographic Primitives. We instantiate the hash function H with BLAKE3[4] and pseudorandom generator G with AES. We re-implement OT extension in Rust following the `emp-toolkit` library[5].

Instantiation of OPRF. We based our OPRF implementation on the VOLE-PSI framework [RR22, RS21], which consists of two building blocks, a vector oblivious linear evaluation (VOLE) protocol [BCGI18, BCG+19, BCG+20] and oblivious key-value stores (OKVS) [GPR+21]. We instantiate VOLE with the state-of-the-art construction by Weng et al. [WYKW21] based on learning parity with noise (LPN) assumptions, and we re-implement the construction in Rust. We include the LPN parameters in Table 5 (Appendix C) and refer readers to [WYKW21] for more details We instantiate OKVS with the state-of-the-art construction by Bienstock et al. [BPSY23].

Cuckoo Hashing. For cuckoo hashing parameters, we use 3 hash functions and no stash. Following [CLR17], we use a lower bound $\lambda = 124\eta - 145$, where λ is the statistical parameter and η is the rate of the hashing table. For all of our experiments, we choose $\eta = 1.5$. The smaller the rate η is, the more favourable for both computation and communication, since it would decrease the number of times we need to call $\mathcal{F}_{\mathsf{dFSS}}$.

Load Balancing. For load balancing between Alice and Bob, as discussed in Section 7.2, we list the chosen set of prefix lengths $\mathcal{L}$ in Table 6 (Appendix C).

7.4 Experimental Results

In this section, we report experimental results and compare with public-key-based approaches [vBP24, GQL+24][6] for fuzzy PSI with ℓ_∞ distance. We did compare with other works [GRS22, GGM24, RRX24] because they do not provide implementations.

We run all experiments on a single AWS EC2 `c5a.x16large` instance with 256 GB RAM. We measure the single-core performance in a LAN setting with network RTT latency of 0.02 ms and bandwidth of 10 Gbps. We consider the balanced setting where Alice has N structured sets and Bob has N unstructured points. Table 4 shows the communication cost and running time for set sizes $N = 2^8, 2^{12}, 2^{16}$, various radii of the ℓ_∞ ball, and dimensions $d = 2, 3, 4$.

As discussed in Section 7.1, the computation and communication costs of [vBP24, GQL+24] grow linearly in the diameter δ, whereas ours only grow polylogarithmically. This exponential gap is evident in the results shown in Table 4. For instance, for set size 2^{16} and radius 10, our protocol achieves a speedup of 5.17× for $d = 2$ and 2.63× for $d = 3$ compared to prior work. As the radius increases to 250, the advantage grows to 27.0× for $d = 2$ and 13.2× for $d = 3$. We illustrate the exponential gap in Figure 8.

[3] Our implementation is available at https://anonymous.4open.science/r/SA-PSI-798C/README.md

[4] https://github.com/BLAKE3-team

[5] https://github.com/emp-toolkit

[6] The implementation for [vBP24] is available at https://github.com/sihangpu/fuzzy_PSI. The implementation [GQL+24] is available at https://github.com/ql70ql70/Fuzzy-Private-Set-Intersection-from-Fuzzy-Mapping/tree/main.

Set size	Radius	Protocol	$d=2$		$d=3$		$d=4$	
			Comm (MB)	Time (s)	Comm (MB)	Time (s)	Comm (MB)	Time (s)
2^8	10	[vBP24]	**1.03**	**0.87**	**1.55**	**1.49**	**2.06**	**2.44**
		[GQL+24]	7.52	2.50	11.2	3.51	14.8	4.87
		Ours	3.02	1.35	5.84	1.72	19.9	5.30
	30	[vBP24]	**3.00**	2.29	**4.50**	3.83	**6.00**	**6.02**
		[GQL+24]	21.4	5.89	32.1	8.64	42.7	11.6
		Ours	3.51	**1.41** 1.62×	9.72	**2.07** 1.85×	54.6	8.80
	60	[vBP24]	5.95	4.45	**8.92**	7.32	**11.9**	**11.4**
		[GQL+24]	42.3	10.6	63.4	17.0	84.4	22.2
		Ours	**3.69** 1.61×	**1.49** 2.98×	10.0	**2.35** 3.11×	55.0	13.6
	120	[vBP24]	11.9	8.67	17.8	14.3	**23.7**	22.2
		[GQL+24]	84.1	21.6	126	32.6	168	44.3
		Ours	**4.31** 2.75×	**1.59** 5.45×	**16.2** 1.09×	**2.90** 4.91×	127	**20.3** 1.09×
	250	[vBP24]	24.6	17.9	36.9	29.2	**49.3**	45.0
		[GQL+24]	174	44.6	261	68.1	349	92.0
		Ours	**4.49** 5.49×	**1.77** 10.1×	**16.5** 2.23×	**3.48** 8.38×	128	**28.5** 1.58×
2^{12}	10	[vBP24]	**16.5**	14.0	**24.8**	24.1	**33.0**	**39.5**
		[GQL+24]	120	39.4	179	60.6	237	79.6
		Ours	29.0	**3.85** 3.64×	74.6	**9.65** 2.49×	298	62.5
	30	[vBP24]	48.0	36.7	**72.0**	61.9	**96.0**	**97.2**
		[GQL+24]	343	101	513	144	683	198
		Ours	**37.3** 1.28×	**4.97** 7.39×	137	**15.5** 3.98×	854	123
	60	[vBP24]	95.2	71.0	143	119	**190**	**183**
		[GQL+24]	677	179	1,014	284	1,351	363
		Ours	**40.0** 2.37×	**6.66** 10.6×	**141** 1.01×	**19.7** 6.03×	859	199
	120	[vBP24]	190	139	284	229	**379**	355
		[GQL+24]	1,345	362	2,016	547	2,687	726
		Ours	**49.9** 3.80×	**8.61** 16.1×	**241** 1.18×	**28.9** 7.91×	2,025	**292** 1.21×
	250	[vBP24]	394	288	591	478	**788**	723
		[GQL+24]	2,793	750	4,188	1,129	5,583	1,470
		Ours	**52.6** 7.48×	**12.6** 22.8×	**245** 2.41×	**37.5** 12.7×	2,031	**436** 1.66×
2^{16}	10	[vBP24]	**264**	226	**396**	385	**528**	**640**
		[GQL+24]	1,924	673	2,859	1,004	3,796	1,328
		Ours	436	**43.7** 5.17×	1,165	**146** 2.63×	4,737	992
	30	[vBP24]	768	593	**1,151**	1,004	**1,535**	**1,562**
		[GQL+24]	5,488	1,645	8,205	2,390	10,924	3,254
		Ours	**568** 1.35×	**61.1** 9.71×	2,162	**236** 4.26×	13,633	1,992
	60	[vBP24]	1,523	1,152	2,284	1,892	**3,045**	**2,974**
		[GQL+24]	10,834	3,222	16,224	4,482	21,616	6,042
		Ours	**612** 2.48×	**76.0** 15.2×	**2,229** 1.02×	**294** 6.44×	13,721	3,201
	120	[vBP24]	3,302	2,286	4,549	3,684	**6,065**	5,788
		[GQL+24]	21,526	6,027	32,262	8,868	–	–
		Ours	**770** 4.29×	**111** 20.6×	**3,830** 1.18×	**460** 8.01×	32,381	**4,770** 1.21×
	250	[vBP24]	6,304	4,804	9,456	8,006	**12,608**	12,191
		[GQL+24]	–	–	–	–	–	–
		Ours	**814** 7.74×	**178** 27.0×	**3,896** 2.42×	**607** 13.2×	32,470	**7,089** 1.72×

Table 4: Experimental results for fuzzy PSI with ℓ_∞ distance. Our speedup factor, compared to [vBP24, GQL+24], is highlighted in green. The entries filled with dash – means the programs run out of memory.

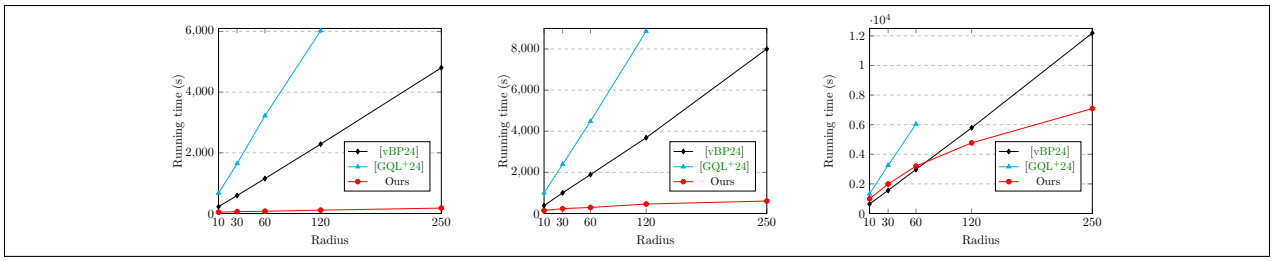

Figure 8: Growth of running time for set size 2^{16} with dimensions $d = 2, 3, 4$.

A similar gap is observed in communication cost. For example, for set size 2^{16} and radius 60, our protocol achieves reduces communication by 2.48× for $d = 2$ and 1.02× for $d = 3$ compared to prior work. As the radius increases to 250, the improvement grows to 7.74× for $d = 2$ and 2.42× for $d = 3$. In fact, for radius 250, our protocol achieves lower communication in all settings except for $d = 4$, for which we estimate an advantage will emerge when the radius reaches 1020.

Finally, since the computation and communication costs of [vBP24,GQL+24] grow quadratically or linearly in the dimension d, whereas ours grow exponentially, our protocol performs better in low-dimensional settings.

Acknowledgments

Dung Bui was supported by the French Agence Nationale de la Recherche (ANR) through the France 2030 project ANR-22-PECY-003 "SecureCompute", and by the DIM Math Innov 2021 program (No. IRIS: 21003816) of the Fondation Sciences Mathématiques de Paris (FSMP), funded by the Île-de-France Region. Gayathri Garimella, Peihan Miao, and Phuoc Van Long Pham were supported in part by NSF SaTC Award 2247352, Meta Research Award, Google Research Scholar Award, and Amazon Research Award.

References

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

[BBC+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*, pages 762–776. IEEE Computer Society Press, May 2021.

[BC23] Dung Bui and Geoffroy Couteau. Improved private set intersection for sets with small entries. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 190–220. Springer, Cham, Switzerland, May 2023.

[BCG+19] 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, Cham, Switzerland, August 2019.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, Cham, Switzerland, August 2020.

[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 Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 871–900. Springer, Cham, Switzerland, October 2021.

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

[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.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Berlin, Heidelberg, Germany, April 2015.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.

[BGIK22] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable distributed point functions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 121–151. Springer, Cham, Switzerland, August 2022.

[BMRR21] Saikrishna Badrinarayanan, Peihan Miao, Srinivasan Raghuraman, and Peter Rindal. Multi-party threshold private set intersection with sublinear communication. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 349–379. Springer, Cham, Switzerland, May 2021.

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

[CFR23] Anrin Chakraborti, Giulia Fanti, and Michael K. Reiter. Distance-aware private set intersection. In *USENIX Security 2023*, pages 319–336. USENIX Association, August 2023.

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

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

[CLO24] Wutichai Chongchitmate, Steve Lu, and Rafail Ostrovsky. Approximate PSI with near-linear communication. Cryptology ePrint Archive, Report 2024/682, 2024.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1243–1255, 2017.

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

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

[DPT20] Thai Duong, Duong Hieu Phan, and Ni Trieu. Catalic: Delegated PSI cardinality with applications to contact tracing. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 870–899. Springer, Cham, Switzerland, December 2020.

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

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

[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, Berlin, Heidelberg, Germany, February 2005.

[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, Berlin, Heidelberg, Germany, May 2004.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, August 1986.

[GGM24] Gayathri Garimella, Benjamin Goff, and Peihan Miao. Computation efficient structure-aware PSI from incremental function secret sharing. In *Advances in Cryptology – CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 309–345. Springer, Cham, Switzerland, August 2024.

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

[GQL+24] Ying Gao, Lin Qi, Xiang Liu, Yuanchao Luo, and Longxin Wang. Efficient fuzzy private set intersection from fuzzy mapping. In *Advances in Cryptology – ASIACRYPT 2024, Part VI*, LNCS, pages 36–68. Springer, Singapore, Singapore, December 2024.

[GRS22] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. Structure-aware private set intersection, with applications to fuzzy matching. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 323–352. Springer, Cham, Switzerland, August 2022.

[GRS23] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. Malicious secure, structure-aware private set intersection. In *CRYPTO 2023, Part I*, LNCS, pages 577–610. Springer, Cham, Switzerland, August 2023.

[GS19] Satrajit Ghosh and Mark Simkin. The communication complexity of threshold private set intersection. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 3–29. Springer, Cham, Switzerland, August 2019.

[GYW+23] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-tree: Halving the cost of tree expansion in COT and DPF. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 330–362. Springer, Cham, Switzerland, April 2023.

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

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

[HOS17] Per A. Hallgren, Claudio Orlandi, and Andrei Sabelfeld. PrivatePool: Privacy-preserving ridesharing. In Boris Köpf and Steve Chong, editors, *CSF 2017 Computer Security Foundations Symposium*, pages 276–291. IEEE Computer Society Press, 2017.

[IKN+19] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Mariana Raykova, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. On deploying secure computing commercially: Private intersection-sum protocols and their business applications. Cryptology ePrint Archive, Report 2019/723, 2019.

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

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 54–70. Springer, Berlin, Heidelberg, Germany, August 2013.

[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.

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

[Lin16] Yehuda Lindell. How to simulate it - a tutorial on the simulation proof technique. Cryptology ePrint Archive, Paper 2016/046, 2016.

[LZQ23] Feng-Hao Liu, En Zhang, and Leiyong Qin. Efficient multiparty probabilistic threshold private set intersection. In *ACM CCS 2023*, pages 2188–2201. ACM Press, November 2023.

[Pag08] Rasmus Pagh. *Cuckoo Hashing*, pages 212–215. Springer US, Boston, MA, 2008.

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

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

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Berlin, Heidelberg, Germany, December 2009.

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

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

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

[Rab05] Michael O Rabin. How to exchange secrets with oblivious transfer. *Cryptology ePrint Archive*, 2005.

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

[RRX24] David Richardson, Mike Rosulek, and Jiayu Xu. Fuzzy PSI via oblivious protocol routing. Cryptology ePrint Archive, Report 2024/1642, 2024.

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

[UCK+21] Erkam Uzun, Simon P. Chung, Vladimir Kolesnikov, Alexandra Boldyreva, and Wenke Lee. Fuzzy labeled private set intersection with applications to private real-time biometric search. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 911–928. USENIX Association, August 2021.

[vBP24] Aron van Baarsen and Sihang Pu. Fuzzy private set intersection with large hyperballs. In *Advances in Cryptology – EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 340–369. Springer, Cham, Switzerland, June 2024.

[WY23] Mingli Wu and Tsz Hon Yuen. Efficient unbalanced private set intersection cardinality and user-friendly privacy-preserving contact tracing. In *USENIX Security 2023*, pages 283–300. USENIX Association, August 2023.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1074–1091. IEEE, 2021.

A Primitives and Definitions

Ball of diameter δ in ℓ_∞ metric space. Let $\vec{x} = (x_1, \ldots, x_d)$ be a point in a d-dimensional space. The ℓ_∞ norm of $\vec{x}$ is defined as $|\vec{x}|_\infty := \max_i |x_i|$. A **ball** consists of the set of points within some distance of a center point. The ball of diameter δ (or radius $\frac{\delta}{2}$) centered at $\vec{x}$ is defined as $\mathcal{B}(\vec{x}, \frac{\delta}{2}) := \{\vec{y} \mid |\vec{x} - \vec{y}|_\infty \leq \frac{\delta}{2}\}$. Under the ℓ_∞ norm, a ball $\mathcal{B}(x, \frac{\delta}{2})$ is a polytope with $2d$ faces, *i.e.*, the intersection of $2d$ half-spaces. Namely, the ball $\mathcal{B}(\mathbf{0}, \frac{\delta}{2})$ centered at the origin is the intersection of all half-spaces of the form $\pm x_i \leq \frac{\delta}{2}$ for every i and every choice of sign.

A.1 Cryptographic Primitives

Pseudorandom Generator. A pseudorandom generator (PRG) is a deterministic function $G : \{0,1\}^\kappa \to \{0,1\}^{\gamma(\kappa)}$ where $G(x)$ for a randomly sampled $x \leftarrow \{0,1\}^\kappa$ is computationally indistinguishable from a string randomly sampled from $\{0,1\}^{\gamma(\kappa)}$ for any PPT adversary.

A.2 Secure Two-Party Computation

Semi-Honest Security. We define secure two-party computation protocols, in the presence of semi-honest adversaries, using the *simulation proof* technique [Lin16]. Parties P_0 with input x_0 and P_1 with input x_1 run protocol Π to learn the output of a function $f(x_0, x_1)$ where party P_i learns $f_i(x_0, x_1)$. Party P_i's $\mathsf{View}_i(x_0, x_1)$ during an honest execution consists of her private input x_i, privately chosen randomness and the transcript of the protocol. Let $\mathsf{Out}_i(x_0, x_1)$ be P_i's output from protocol Π. We say that protocol Π securely realizes a functionality f if there exists a PPT simulator Sim for both parties and for all possible inputs x_0, x_1 such that for $i \in \{0,1\}$,

$$(\mathsf{Sim}(1^\kappa, i, x_i, f_i(x_0, x_1)), f_{1-i}(x_0, x_1)) \cong_\kappa (\mathsf{View}_i(x_0, x_1), \mathsf{Out}_{1-i}(x_0, x_1)) .$$

A.3 Building Blocks

In our constructions, we make use of three building blocks including Oblivious Transfer (OT) [Rab05], Oblivious Pseudorandom Function (OPRF) [FIPR05], and Cuckoo Hashing Schemes [Pag08].

Oblivious Transfer. Oblivious Transfer (OT) [Rab05] is a fundamental two-party cryptographic protocol. In a single OT instance, the sender provides a pair of messages (m_0, m_1), and the receiver learns exactly one of them m_b, based on his selection bit $b \in \{0,1\}$, without revealing b to the sender. The ideal functionality for OT instances is formally described in Figure 9. While a single OT typically relies on public-key operations, a large batch of OTs can be implemented efficiently using only $\mathcal{O}(\kappa)$ public-key operations followed by symmetric-key operations through the use of *OT extension* techniques [IKNP03, KK13, ALSZ13].

Parameter: message length ℓ

Functionality:

1. Receive $m_0, m_1 \in \{0,1\}^\ell$ from the sender and $b \in \{0,1\}$ from the receiver.
2. Send output m_b to the receiver.

Figure 9: Ideal functionality $\mathcal{F}_{\mathsf{OT}}$ for oblivious transfer.

Oblivious Pseudo-random Function. Oblivious Pseudo-random Function (OPRF) [FIPR05] is a secure two-party computation protocol between a sender and a receiver, where the receiver holds an input set X. The protocol allows the sender to learn a PRF key (that enables evaluation on any input) and the receiver to learn the evaluation of the PRF on all $x \in X$. The ideal functionality of OPRF is defined in Figure 10. Constructing OPRF is a common approach to designing a PSI protocol [CM20, KKRT16, RR22, RS21], recently OPRF-based PSI [RR22, RS21] is significantly efficient by leveraging the efficiency of VOLE [BCGI18, BCG+19, BCG+20].

Parameters:

- a sender and a receiver with input set X, size of the receiver's input set $|X| = n$
- output bit length out

Functionality:

1. Upon receiving (sender) from Sender and $(\mathsf{receiver}, X)$ from Receiver, the functionality samples a random function $\mathsf{PRF} : \{0,1\}^* \to \{0,1\}^{\mathsf{out}}$, and returns $O = \{\mathsf{PRF}(x) \mid x \in X\}$ to Receiver.
2. Subsequently, upon receiving (sender, y) from Sender, the functionality returns $\mathsf{PRF}(y)$ to Sender.

Figure 10: Ideal functionality $\mathcal{F}_{\mathsf{oprf}}$ for (two-party) Oblivious Pseudorandom Function.

B Security Proofs

B.1 Proof of Theorem 1

Theorem. *For a set family $\mathcal{S} \subseteq \underbrace{2^{\mathcal{U}} \times \cdots \times 2^{\mathcal{U}}}_{d}$ where $\mathcal{U} = \{0,1\}^u$ and every set in $\mathcal{S}$ is a disjoint union of at most w prefix sets, given an ideal functionality for distributed function secret sharing $\mathcal{F}_{\mathsf{dFSS}}$ for $\mathsf{riSMF}_{\vec{S},\beta}$ (where $\vec{S} \in \mathcal{S}$ and $\beta \in \{0,1\}^\kappa$) and a correlation robust hash $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{\lambda+\log|N_A|+\log|N_B|+\log w+d\log u}$, the protocol in Figure 3 realizes $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$ (Figure 2) in the $\mathcal{F}_{\mathsf{dFSS}}$-hybrid model, in the presence of semi-honest adversaries.*

Proof. We describe a simulation-based proof, where we will construct a simulator Sim that simulates the view of a corrupted party.

Alice is corrupted. First, we consider the case where Alice is corrupt. Alice's view in the protocol consists of the output from the $\mathcal{F}_{\mathsf{dFSS}}$ ideal functionality and a set Y of N_B hash evaluations received from Bob. Below, we present a sequence of hybrids starting from the real execution of the protocol and ending with a simulation in the ideal world, which is independent of Bob's input.

Hybrid 0. This is the real protocol execution with an honest Bob. For the sake of the proof, we will express the set Y from Bob as the disjoint union of the following sets.

$$Y = \left\{ \mathsf{H}\left(\vec{y}' \| i; [\![\mathbf{r}_i^{(\vec{y}')}]\!]_b\right) \;\middle|\; \begin{array}{l} i \in [N_A], \vec{y} = (y_1, \ldots, y_d) \in S_B, \\ \ell_1 \in [u], \ldots, \ell_d \in [u], \\ \vec{y}' := ((y_1)_{[1:\ell_1]}, \ldots, (y_d)_{[1:\ell_d]}) \end{array} \right\}$$
$$= Y_1 \dot{\cup} Y_2 \dot{\cup} Y_3 \dot{\cup} Y_4.$$

Y_1, Y_2, Y_3 and Y_4 are defined as follows:

$$Y_1 = \left\{ \mathsf{H}\left(\vec{y}' \| i; [\![\mathbf{r}_i^{(\vec{y}')}]\!]_a\right) \;\middle|\; \begin{array}{l} i \in [N_A], S_A^{(i)} = \dot{\bigcup}_{j\in[w']} \mathsf{PreS}_{\vec{x}^j}, \vec{x}^j = (x_1^j, \ldots, x_d^j), \\ \vec{y} = (y_1, \ldots, y_d) \in \mathsf{PreS}_{\vec{x}^j} \cap S_B, \\ \ell_1 \in [|x_1^j|, u], \ldots, \ell_d \in [|x_d^j|, u], \\ \vec{y}' := ((y_1)_{[1:\ell_1]}, \ldots, (y_d)_{[1:\ell_d]}) \end{array} \right\}$$

$$Y_2 = \left\{ \mathsf{H}\left(\vec{y}' \| i; [\![\mathbf{r}_i^{(\vec{y}')}]\!]_a \oplus r\right) \;\middle|\; \begin{array}{l} i \in [N_A], S_A^{(i)} = \dot{\bigcup}_{j\in[w']} \mathsf{PreS}_{\vec{x}^j}, \vec{x}^j = (x_1^j, \ldots, x_d^j), \\ \vec{y} = (y_1, \ldots, y_d) \in \mathsf{PreS}_{\vec{x}^j} \cap S_B, \\ (\ell_1, \ldots, \ell_d) \in [u]^d \text{ where } \ell_1 \in [|x_1^j - 1|] \vee \cdots \vee \ell_d \in [|x_d^j - 1|], \\ \vec{y}' := ((y_1)_{[1:\ell_1]}, \ldots, (y_d)_{[1:\ell_d]}) \end{array} \right\}$$

$$Y_3 = \left\{ \mathsf{H}\left(\vec{y}' \| i; [\![\mathbf{r}_i^{(\vec{y}')}]\!]_a \oplus r\right) \;\middle|\; \begin{array}{l} i \in [N_A], \vec{y} = (y_1, \ldots, y_d) \in S_B \setminus \vec{S}_A^{(i)}, \\ \ell_1 \in [u], \ldots, \ell_d \in [u], \\ \vec{y}' := ((y_1)_{[1:\ell_1]}, \ldots, (y_d)_{[1:\ell_d]}) \end{array} \right\}$$

and Y_4 is the set of dummy items added to ensure $|Y| = N_B \cdot u^d$.

Hybrid 1. This hybrid is same as Hybrid 0 except that we replace hash evaluations in Y_2, Y_3 with uniformly sampled strings of length $\lambda + \log|N_A| + \log|N_B| + \log w + d\log u$. Now, Bob's message Y to Alice looks like

$$Y = Y_1 \dot{\cup} \{h_1, \ldots, h_{|Y_2|}\} \dot{\cup} \{h'_1, \ldots, h'_{|Y_3|}\} \dot{\cup} Y_4$$

where each $h_i, h'_i \leftarrow\$ \{0,1\}^{\lambda+\log|N_A|+\log|N_B|+\log w+d\log u}$. This hybrid is indistinguishable from the previous hybrid. First, by the property of $\mathcal{F}_{\mathsf{dFSS}}$ we can write hash evaluations in Y_2 and Y_3 as $\mathsf{H}\left(\vec{y}' \| i; [\![\mathbf{r}_{\vec{y}'}^{(i)}]\!]_a \oplus r\right)$, where $r = (\{0,1\}^\kappa)^d$ and has κ bits on entropy . Next, we utilize the hamming correlation robust hash property (Definition 2). We can claim that the hash outputs in Y_2 and Y_3 are jointly pseudo random, since each hash value is computed on a unique prefix $\vec{y}' \| i$ and r has κ bits of randomness.

Simulator. Sim for corrupted Alice can be constructed as follows.

- Corrupt Alice samples $\left\{[\![\mathbf{r}_i^{(\vec{x})}]\!]_b\right\}_{\vec{x}\in\left(\bigcup_{\ell=1}^u \{0,1\}^\ell\right)^d}$ for each $i \in [N_A]$ and sends it to the ideal functionality $\mathcal{F}_{\mathsf{dFSS}}$. Sim forwards $\left\{[\![\mathbf{r}_i^{(\vec{x})}]\!]_b\right\}_{\vec{x}\in\left(\bigcup_{\ell=1}^u \{0,1\}^\ell\right)^d}$ to Alice as output of $\mathcal{F}_{\mathsf{dFSS}}$ ideal functionality.
- Sim sends S_A to $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$ and learns $I = S_A \cap S_B$. From I, Sim defines the set Y as follows.
  - Let $Y_1 = I$ and Y_2, Y_3 and Y_4 are sampled uniformly with the same output length as the output of hash function H, such that $|Y| = N_B \cdot u^d$.
- Sim sends Y to Alice on behalf of Bob.

Bob is corrupted. In the protocol, Bob's view consists solely of the output from the $\mathcal{F}_{\mathsf{dFSS}}$ functionality. The simulator is defined as follows:

- Corrupt Bob samples $\left\{[\![\mathbf{r}_i^{(\vec{x})}]\!]_b\right\}_{\vec{x}\in\left(\bigcup_{\ell=1}^{u}\{0,1\}^\ell\right)^d}$ for each $i \in [N_A]$ and sends it to the ideal functionality $\mathcal{F}_{\mathsf{dFSS}}$. Sim forwards $\left\{[\![\mathbf{r}_i^{(\vec{x})}]\!]_b\right\}_{\vec{x}\in\left(\bigcup_{\ell=1}^{u}\{0,1\}^\ell\right)^d}$ to Bob as output of $\mathcal{F}_{\mathsf{dFSS}}$ ideal functionality.
- Sim sends S_B to $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$.

Correctness. We consider Alice's simulated view since it is indistinguishable from her view in the honest execution of the protocol. For every input $\vec{x} \in S_B \cap \mathsf{PreS}_{\vec{x}^j}$ in the intersection (over all choices of $i \in [N_A], S_A^{(i)} = \dot{\bigcup}_{j\in[w']}\mathsf{PreS}_{\vec{x}^j}$, $\vec{x}^j = (x_1^j, \ldots, x_d^j)$), the simulator includes the following hash values in the message to Alice:

$$\left\{\mathsf{H}\left(\vec{x}'\|i; [\![\mathbf{r}_i^{(\vec{x}')}]\!]_a\right)\right) \;\middle|\; \begin{array}{l} \ell_1 \in [|x_1^j| : u], \ldots, \ell_d \in [|x_d^j| : u] \\ \vec{x}' := ((x_1)_{[1:\ell_1]}, \ldots, (x_d)_{[1:\ell_d]}) \end{array} \right\}$$

Alice can uniquely compute and recognize hash values from Bob's message and will correctly include every such element $\vec{x}$ in the output. Further, the chance that Alice includes an element $\vec{x} \in S_A \setminus S_B$ is the probability that she finds the hash output of prefix of $\vec{x}$ for some i' in Bob's message, that is, $\mathsf{H}\left(\vec{x}'\|i'; [\![\mathbf{r}_{i'}^{(\vec{x}')}]\!]\right)\right) \in Y$, where $\vec{x} \in \mathsf{PreS}_{\vec{x}'}$. By union bound, the probability that Alice finds a value matching for a specific prefix $\mathsf{PreS}_{\vec{x}'}$ is $|Y|\cdot 2^{-\lambda-\log|N_A|-\log|N_B|-\log w-d\log u} = 2^{-\lambda-\log|N_A|-\log w}$. Again, by the union bound over total number of prefixes, the probability that Alice includes a wrong element $\vec{x}$ in her output is less than $w\cdot|N_A|\cdot 2^{-\lambda-\log|N_A|-\log w} = 2^{-\lambda}$, which is statistically negligible.

□

B.2 Proof of Theorem 2

Theorem. *For a set family $\mathcal{S} \subseteq \underbrace{2^{\mathcal{U}'} \times \cdots \times 2^{\mathcal{U}'}}_{d}$ where $\mathcal{U}' = 2\delta$ and every set in $\mathcal{S}$ is a disjoint union of at most w prefix sets, given an ideal functionality for distributed function secret sharing $\mathcal{F}_{\mathsf{dFSS}}$ for $\mathsf{riSMF}_{\vec{S},\beta}$ (where $\vec{S} \in \mathcal{S}$ and $\beta \in \{0,1\}^\kappa$), an ideal functionality for oblivious pseudorandom functions $\mathcal{F}_{\mathsf{oprf}}$, a cuckoo hashing scheme with three hash functions $H_1, H_2, H_3 : \mathcal{U}^d \to [m]$ and no stash, and a correlation robust hash*

$$H : \{0,1\}^* \to \{0,1\}^{\lambda+\log|N_A|+\log w+\log|N_B|+\log 3+d(1+\log\log(2\delta))}$$

, the protocol in Figure 4 realizes $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$ (Figure 2) in the $(\mathcal{F}_{\mathsf{dFSS}}, \mathcal{F}_{\mathsf{oprf}})$-hybrid model, in the presence of semi-honest adversaries.

Proof. **Alice is corrupt.** Now, we will show through a sequence of hybrids how Alice's view can be simulated independent of Bob's input.

Hybrid 0. In the protocol, Alice's receives a set of values Y from Bob.

$$Y = \left\{ \mathsf{H}\left(\vec{y}'\|\vec{o}\|\mathsf{PRF}(t\|\vec{o}); [\![\mathbf{r}_t^{(\vec{y}')}]\!]_b\right) \;\middle|\; \begin{array}{l} \vec{y} \in S_B, \vec{y} \in \mathsf{Univ}_{\vec{o}}, \vec{y}.\mathsf{Shift}(\vec{o}) = (y_1, \ldots, y_d) \\ t = \mathsf{H}_\tau(\vec{o}) \text{ where } \tau = \{1,2,3\}, \\ \ell_1 \in [\log(2\delta)], \ldots, \ell_d \in [\log(2\delta)], \\ \vec{y'} := ((y_1)_{[1:\ell_1]}, \ldots, (y_d)_{[1:\ell_d]}) \end{array} \right\}$$

We write the set $Y = Y_1 \dot\cup Y_2 \dot\cup Y_3 \dot\cup Y_4 \dot\cup Y_5 \dot\cup Y_6$ as the disjoint union of sets, where each term can be expressed as follows.

$$Y_1 = \left\{ \mathsf{H}\left(\vec{y'}\|\vec{o}^{(i)}\|\mathsf{PRF}(t\|\vec{o}^{(i)}); [\![\mathbf{r}_t^{(\vec{y'})}]\!]_a\right) \;\middle|\; \begin{array}{l} i \in [N_A], \vec{S}_A^{(i)}.\mathsf{Shift}(\vec{o}^{(i)}) = \dot\bigcup_{j\in[w']} \mathsf{PreS}_{\overrightarrow{x^j}}, \\ \overrightarrow{x^j} = (x_1^j, \ldots, x_d^j), \mathsf{T}[t] = (\vec{o}^{(i)}, \vec{S}_A^{(i)}), \\ \vec{y} \in S_B, \vec{y} \in \mathsf{Univ}_{\vec{o}^{(i)}}, \vec{y}.\mathsf{Shift}(\vec{o}^{(i)}) = (y_1, \ldots, y_d) \in \mathsf{PreS}_{\overrightarrow{x^j}}, \\ \ell_1 \in [|x_1^j| : \log(2\delta)], \ldots, \ell_d \in [|x_d^j| : \log(2\delta)], \\ \vec{y'} := ((y_1)_{[1:\ell_1]}, \ldots, (y_d)_{[1:\ell_d]}) \end{array} \right\}$$

$$Y_2 = \left\{ \mathsf{H}\left(\vec{y'}\|\vec{o}^{(i)}\|\mathsf{PRF}(t\|\vec{o}^{(i)}); [\![\mathbf{r}_t^{(\vec{y'})}]\!]_a \oplus r\right) \;\middle|\; \begin{array}{l} i \in [N_A], \vec{S}_A^{(i)}.\mathsf{Shift}(\vec{o}^{(i)}) = \dot\bigcup_{j\in[w']} \mathsf{PreS}_{\overrightarrow{x^j}}, \\ \overrightarrow{x^j} = (x_1^j, \ldots, x_d^j), \mathsf{T}[t] = (\vec{o}^{(i)}, \vec{S}_A^{(i)}), \\ \vec{y} \in S_B, \vec{y} \in \mathsf{Univ}_{\vec{o}^{(i)}}, \vec{y}.\mathsf{Shift}(\vec{o}^{(i)}) = (y_1, \ldots, y_d) \in \mathsf{PreS}_{\overrightarrow{x^j}}, \\ \ell_1 \in [|x_1^j - 1|] \vee \ldots \vee \ell_d \in [|x_d^j - 1|], (\ell_1, \ldots, \ell_d) \in [\log(2\delta)]^d \\ \vec{y'} := ((y_1)_{[1:\ell_1]}, \ldots, (y_d)_{[1:\ell_d]}) \end{array} \right\}$$

$$Y_3 = \left\{ \mathsf{H}\left(\vec{y'}\|\vec{o}^{(i)}\|\mathsf{PRF}(t\|\vec{o}^{(i)}); [\![\mathbf{r}_t^{(\vec{y'})}]\!]_a \oplus r\right) \;\middle|\; \begin{array}{l} i \in [N_A], \vec{S}_A^{(i)}.\mathsf{Shift}(\vec{o}^{(i)}) = \dot\bigcup_{j\in[w']} \mathsf{PreS}_{\overrightarrow{x^j}}, \\ \overrightarrow{x^j} = (x_1^j, \ldots, x_d^j), \mathsf{T}[t] = (\vec{o}^{(i)}, \vec{S}_A^{(i)}), \\ \vec{y} \in S_B, \vec{y} \in \mathsf{Univ}_{\vec{o}^{(i)}}, \vec{y}.\mathsf{Shift}(\vec{o}^{(i)}) = (y_1, \ldots, y_d) \notin \mathsf{PreS}_{\overrightarrow{x^j}}, \\ \ell_1 \in [\log(2\delta)], \ldots, \ell_d \in [\log(2\delta)], \\ \vec{y'} := ((y_1)_{[1:\ell_1]}, \ldots, (y_d)_{[1:\ell_d]}) \end{array} \right\}$$

$$Y_4 = \left\{ \mathsf{H}\left(\vec{y'}\|\vec{o}^{(i)}\|\mathsf{PRF}(t'\|\vec{o}^{(i)}); [\![\mathbf{r}_t^{(\vec{y'})}]\!]_a \oplus r\right) \;\middle|\; \begin{array}{l} i \in [N_A], \vec{S}_A^{(i)}.\mathsf{Shift}(\vec{o}^{(i)}) = \dot\bigcup_{j\in[w']} \mathsf{PreS}_{\overrightarrow{x^j}}, \overrightarrow{x^j} = (x_1^j, \ldots, x_d^j), \\ \mathsf{T}[\mathsf{H}_\tau(\vec{o}^{(i)})] = \mathsf{T}[t'] \neq (\vec{o}^{(i)}, \vec{S}_A^{(i)}) \text{ for } \tau = \{1,2,3\} \\ \vec{y} \in S_B, \vec{y} \in \mathsf{Univ}_{\vec{o}^{(i)}}, \\ \ell_1 \in [\log(2\delta)], \ldots, \ell_d \in [\log(2\delta)], \\ \vec{y'} := ((y_1)_{[1:\ell_1]}, \ldots, (y_d)_{[1:\ell_d]}) \end{array} \right\}$$

$$Y_5 = \left\{ \mathsf{H}\left(\vec{y'}\|\vec{o}^{(i)}\|\mathsf{PRF}(t'\|\vec{o}^{(i)}); [\![\mathbf{r}_t^{(\vec{y'})}]\!]_a \oplus r\right) \;\middle|\; \begin{array}{l} i \in [N_A], \vec{y} \in S_B, \vec{y} \in \mathsf{Univ}_{\vec{o}^{(i)}}, \\ \vec{y}.\mathsf{Shift}(\vec{o}^{(i)}) = (y_1, \ldots, y_d) \notin \vec{S}_A^{(i)}.\mathsf{Shift}(\vec{o}^{(i)}), \\ \mathsf{H}_\tau(\vec{o}^{(i)}) = t' \text{ for } \tau = \{1,2,3\} \\ \ell_1 \in [\log(2\delta)], \ldots, \ell_d \in [\log(2\delta)], \\ \vec{y'} := ((y_1)_{[1:\ell_1]}, \ldots, (y_d)_{[1:\ell_d]}) \end{array} \right\}$$

and Y_6 consists of dummy pseudorandom strings of length equal to the hash outputs, such that $|Y| = 3 \cdot N_B \cdot (2 \cdot \log(2\delta))^d$. In the protocol, every set $S_A^{(i)}$ in Alice's input is uniquely mapped to the origin $\vec{o}^{(i)}$ of a mini-universe and a specific bin position in the cuckoo hash table $\mathsf{T}[t] = (\vec{o}^{(i)}, \vec{S}_A^{(i)})$, such that $\vec{S}_A^{(i)}.\mathsf{Shift}(\vec{o}^{(i)}) \subseteq \mathsf{Univ}_{\vec{o}^{(i)}}$. When Bob evaluates on a point $\vec{y} \in S_B$, he considers all the (bounded set of) origins (and in turn mini-universes) and 3 potential bin positions that contain his input and sends hash evaluations on all prefix combinations of his input vector. We categorized Bob's message as follows

- Y_1, Y_2 set of evaluations for Bob's inputs that belongs to Alice's structure contained in the same origin, and match the position in cuckoo table. Y_1 consists of all those prefix evaluations that are in Alice's prefix set and Y_2 consists of all evaluations outside that.
- Y_3 set of evaluations of Bob's inputs (and all prefixes) that belong to an origin that consists of Alice's structure and match the position in cuckoo hash table chosen by Alice. However, the point itself lies outside the structure.
- Y_4 set of evaluations of Bob's inputs (and all prefixes) that belong to an origin that consists of Alice's structure but does not match with Alice's choice of position in cuckoo hash table.
- Y_5 set of hash evaluations of Bob's inputs (and all prefixes) that belong to an origin that does not contain any of Alice's inputs.

Hybrid 1. We modify the real execution of the protocol in this hybrid. We replace all the hash outputs in Y_2, Y_3 and Y_5 with uniformly sampled strings of length $\lambda + \log|N_A| + \log w + \log|N_B| + \log 3 + d(1 + \log\log(2\delta))$. Using the property of $\mathcal{F}_{\mathsf{dFSS}}$ we can write hash evaluations in Y_2, Y_3, Y_5 as $\mathsf{H}\left(\vec{y}'\|\vec{o}^{(i)}\|\mathsf{PRF}(t\|\vec{o}^{(i)}); [\![\mathbf{r}_t^{(\vec{y}')}]\!]_a \oplus r\right)$, such that $r = (\{0,1\}^\kappa)^d$ and has κ bits of entropy. Further, all these hash evaluations are computed on unique prefixes $\vec{y}'\|\vec{o}^{(i)}\|\mathsf{PRF}(t\|\vec{o}^{(i)})$. Therefore, by the hamming correlation robust hash definition (Definition 2) all the hash outputs in (Y_2, Y_3, Y_5) are jointly pseudo random and can be replaced by uniformly random strings.

Hybrid 2. This hybrid is identical to previous hybrid, except that we replace all the hash outputs in Y_4 with uniformly random strings of length $\lambda + \log|N_A| + \log w + \log|N_B| + \log 3 + d(1 + \log\log(2\delta))$. We argue that this is indistinguishable from previous hybrid because Alice will compute PRF on a distinct input from Bob for the same prefix. As a result the hash evaluations from Bob will look pseudo random to Alice.

Simulator. We can construct a simulator for a corrupt Alice as follows.

- Corrupt Alice samples $\left\{[\![\mathbf{r}_t^{(\vec{x})}]\!]_a\right\}_{\vec{x}\in\left(\bigcup_{\ell=1}^{\log(2\delta)}\{0,1\}^\ell\right)^d}$ for each $t \in [m]$ and sends it to the ideal functionality $\mathcal{F}_{\mathsf{dFSS}}$. Sim forwards $\left\{[\![\mathbf{r}_t^{(\vec{x})}]\!]_a\right\}_{\vec{x}\in\left(\bigcup_{\ell=1}^{\log(2\delta)}\{0,1\}^\ell\right)^d}$ to Alice as output of $\mathcal{F}_{\mathsf{dFSS}}$ ideal functionality.
- Sim sends S_A to $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$ and learns $I = S_A \cap S_B$. From I, Sim defines the set Y as follows.
  - Let $Y_1 = I$ and Y_2, Y_3, Y_4, Y_5 and Y_6 are sampled uniformly with the same output length as the output of hash function H, such that $|Y| = 3 \cdot N_B \cdot (2 \cdot \log(2\delta))^d$.
- Sim sends Y to Alice on behalf of Bob.

Bob is corrupt. We can construct a simulator for a corrupt Bob as follows.

- Bob samples $\left\{[\![\mathbf{r}_t^{(\vec{y})}]\!]_b\right\}_{\vec{y}\in\left(\bigcup_{\ell=1}^{\log(2\delta)}\{0,1\}^\ell\right)^d}$ for each bin $t \in [m]$ in the hash table and sends it to the ideal functionality $\mathcal{F}_{\mathsf{dFSS}}$. Sim forwards $\left\{[\![\mathbf{r}_t^{(\vec{y})}]\!]_b\right\}_{\vec{y}\in\left(\bigcup_{\ell=1}^{\log(2\delta)}\{0,1\}^\ell\right)^d}$ to Bob as output of $\mathcal{F}_{\mathsf{dFSS}}$ ideal functionality.
- Sim forwards output from the $\mathcal{F}_{\mathrm{oprf}}$ ideal functionality to Bob.
- Sim forwards S_B to $\mathcal{F}_{\mathsf{sa\text{-}PSI}}$.

Correctness. For any input $\vec{x}$ from Alice's set $\vec{S}_A^{(i)}$ which belongs to prefix set $\vec{x} \in \mathsf{PreS}^{\rightarrow}_{x^j}$, that is, $\vec{x}.\mathsf{Shift}(\vec{o}^{(i)}) \in \vec{S}_A^{(i)}.\mathsf{Shift}(\vec{o}^{(i)}) = \dot{\bigcup}_{j\in[w']}\mathsf{PreS}^{\rightarrow}_{x^j}$. And suppose that, Alice's set is placed in bin $t^{(i)}$ in cuckoo hash table $\mathsf{T}[t^{(i)}] = (\vec{o}_A^{(i)}, \vec{S}_A^{(i)})$, we consider the following cases.

- **When $\vec{x} \in S_B$:** set Y_1 will include the following hash values
$$\left\{\mathsf{H}\left(\vec{x}\|\vec{o}^{(i)}\|\mathsf{PRF}(t^{(i)}\|\vec{o}^{(i)}); [\![\mathbf{r}^{(\vec{x})}_{t^{(i)}}]\!]_a\right) \;\middle|\; \begin{array}{l}\ell_1 \in [|x_1^j| : u], \ldots, \ell_d \in [|x_d^j| : u] \\ \vec{x} := ((x_1)_{[1:\ell_1]}, \ldots, (x_d)_{[1:\ell_d]})\end{array}\right\}$$
By collision resistant hash property, Alice can uniquely recognize these hash values and correctly includes $\vec{x}$ in her output.
- **When $\vec{x} \notin S_B$:** Alice includes $\vec{x}$ in her output only when she finds hash of one of her prefix sets $\mathsf{H}\left(\vec{x}^j\|\vec{o}^{(i)}\|\mathsf{PRF}(t^{(i)}\|\vec{o}^{(i)}); [\![\mathbf{r}^{(\vec{x})}_{t^{(i)}}]\!]_a\right)$ in Bob's message. By union bound, the probability of finding such a matching value for a specific prefix over all of Alice's prefixes is
$$2^{-\lambda-\log|N_A|-\log w-\log|N_B|-\log 3-d(1+\log\log(2\delta))} \cdot |Y| \cdot w \cdot |N_A| = 2^{-\lambda}$$
is negligible. Therefore, the probability of wrongly including $\vec{x}$ in the output is upper bounded by $2^{-\lambda}$.

□

B.3 Proof of Theorem 3

Theorem. *The protocol Π_{iDCF} presented in Figure 5 securely realizes $\mathcal{F}_{\mathsf{dFSS}}$ (Figure 1) for incremental distributed comparison functions $\mathsf{iDCF}_{\alpha,\beta}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model, in the presence of semi-honest adversaries.*

Proof. We first show correctness of the protocol through Lemma 1 and Lemma 2.

Lemma 1 (Correctness)**.** *The protocol* Π_{iDCF} *presented in Figure 5 satisfies the correctness guarantee: for each* $x \in \bigcup_{\ell=1}^{u}\{0,1\}^{\ell}$*,*

$$[\![\mathbf{r}^{(x)}]\!]_a \oplus [\![\mathbf{r}^{(x)}]\!]_b = \begin{cases} 0^{\kappa} & \text{if } x < \alpha_{[1:|x|]} \\ \beta & \text{otherwise} \end{cases}.$$

Proof. For each $x \in \bigcup_{\ell=1}^{u}\{0,1\}^{\ell}$, for ease of reading, we abuse the notation ℓ as the bit-length of x in the rest of the proof. We need to prove that when $x < \alpha_{[1:\ell]}$, we have $[\![\mathbf{r}^{(x)}]\!]_a \oplus [\![\mathbf{r}^{(x)}]\!]_b = 0^{\kappa}$, and when $x \geq \alpha_{[1:\ell]}$, we have $[\![\mathbf{r}^{(x)}]\!]_a \oplus [\![\mathbf{r}^{(x)}]\!]_b = \beta$. We first prove the following claim.

Claim. Given $|x| = \ell \in [u]$, we have $c^i_j = \tilde{c}^i_j$ for any $i \in [\ell], j \in [0, 2^{i-1})$ if and only if $x_{[1:i-1]}$ is not a prefix of α ($x_{[1:i-1]} \neq \alpha_{[1:i-1]}$), or j is a prefix of α ($j = \alpha_{[1:i]}$).

As shown in the construction of Figure 5, Alice and Bob obtain $g^i_j = \tilde{g}^i_j$ for any $i \in [\ell], j \in [0, 2^{i-1})$ such that j is not a prefix of α. To prove the above claim, for each $i \in [\ell], j \in [0, 2^{i-1})$, we analyze different cases:

- If $x_{[1:i-1]}$ is not a prefix of α, since $g^i_{x_{[1:i-1]}} = \tilde{g}^i_{x_{[1:i-1]}}$, Alice can compute $\tilde{c}^i_j = C_{j_i}\left(\tilde{g}^i_{x_{[1:i-1]}}\right)$ and obtain:
$$\tilde{c}^i_j = C_{j_i}\left(\tilde{g}^i_{x_{[1:i-1]}}\right) = C_{j_i}\left(g^i_{x_{[1:i-1]}}\right) = c^i_j.$$
- If j is a prefix of α, i.e., $j = \alpha_{[1:i]}$, in the i-th OT, Alice receives $\tilde{\mathbf{c}}^i_{j_i} = \mathbf{c}^i_{j_i}$. Alice can now reconstruct $\tilde{c}^i_j$ as:
$$\tilde{c}^i_j = \tilde{\mathbf{c}}^i_{j_i} \oplus \bigoplus_{j' \in [0,2^{i-1}), j' \neq x_{[1:i-1]}} \tilde{c}^i_{j'\|j_i} = \mathbf{c}^i_{j_i} \oplus \bigoplus_{j' \in [0,2^{i-1}), j' \neq x_{[1:i-1]}} c^i_{j'\|j_i} = c^i_j.$$

When $x_{[1:i-1]}$ is a prefix of α but $j_i \neq \alpha_i$, we show that $\tilde{c}^i_j = c^i_j \oplus \beta$ in two cases:

- If $\alpha_i = 0$, which means $j_i = 1$, we have $\tilde{c}^i_j = c^i_j$ since:
$$\tilde{c}^i_j = \tilde{\mathbf{c}}^i_1 \oplus \bigoplus_{j' \in [0,2^{i-1}), j' \neq x_{[1:i-1]}} \tilde{c}^i_{j'\|1} = \mathbf{c}^i_1 \oplus \bigoplus_{j' \in [0,2^{i-1}), j' \neq x_{[1:i-1]}} c^i_{j'\|1} = c^i_j.$$
- If $\alpha_i = 1$, which means $j_i = 0$, we have $\tilde{c}^i_j = c^i_j \oplus \beta$ since:
$$\tilde{c}^i_j = \tilde{\mathbf{c}^i_0} \oplus \bigoplus_{j' \in [0,2^{i-1}), j' \neq x_{[1:i-1]}} \tilde{c}^i_{j'\|0} = (\mathbf{c}^i_0 \oplus \beta) \oplus \bigoplus_{j' \in [0,2^{i-1}), j' \neq x_{[1:i-1]}} c^i_{j'\|0} = c^i_j \oplus \beta.$$

This concludes the proof for the claim. We now prove Lemma 1 in three cases: $x < \alpha_{[1:\ell]}$, $x = \alpha_{[1:\ell]}$, and $x > \alpha_{[1:\ell]}$.

Case 1: When $x < \alpha_{[1:\ell]}$, let i_0 be the first index such that $x_{i_0} < \alpha_{i_0}$:

- If $i_0 = 1$, it means that $x_1 = 0$ and $\alpha_1 = 1$, and we have $\tilde{c}^1_0 = c^1_0$.
For all $1 < i \leq \ell$, $x_{[1:i]}$ is not a prefix of α, hence $\tilde{c}^i_{x_{[1:i]}} = c^i_{x_{[1:i]}}$. In conclusion, we have $[\![\mathbf{r}^{(x)}]\!]_a \oplus [\![\mathbf{r}^{(x)}]\!]_b = 0^{\kappa}$ since:
$$[\![\mathbf{r}^{(x)}]\!]_b = \tilde{c}^1_0 \oplus \bigoplus_{i=2}^{\ell} \tilde{c}^i_{x_{[1:i]}} = c^1_0 \oplus \bigoplus_{i=2}^{\ell} c^i_{x_{[1:i]}} = [\![\mathbf{r}^{(x)}]\!]_a.$$

- If $i_0 > 1$, we have $x_1 = \alpha_1$, hence $\tilde{c}^1_{x_1} = c^1_{x_1} \oplus \beta$.
  For all $1 < i < i_0$, since $x_{[1:i]}$ is a prefix of α, we have $\tilde{c}^i_{x_{[1:i]}} = c^i_{x_{[1:i]}}$. For all $i_0 < i < \ell$, since $x_{[1:i-1]}$ is not a prefix of α, we also have $\tilde{c}^i_{x_{[1:i]}} = c^i_{x_{[1:i]}}$. The remaining index to analyse is i_0, where $x_{i_0} = 0$ and $\alpha_{i_0} = 1$, hence $\tilde{c}^{i_0}_{j_{[1:i_0]}} = c^{i_0}_{j_{[1:i_0]}} \oplus \beta$. In conclusion, we have $[\![\mathbf{r}^{(x)}]\!]_a \oplus [\![\mathbf{r}^{(x)}]\!]_b = 0^\kappa$ since:

$$\begin{aligned}
[\![\mathbf{r}^{(x)}]\!]_b &= \tilde{c}^1_0 \oplus \bigoplus_{i=2}^{i_0} \tilde{c}^i_{x_{[1:i]}} \oplus \tilde{c}^{i_0}_{j[1:i_0]} \oplus \bigoplus_{i=i_0+1}^{\ell} \tilde{c}^i_{x_{[1:i]}} \\
&= (c^1_0 \oplus \beta) \oplus \bigoplus_{i=2}^{i_0} c^i_{x_{[1:i]}} \oplus (c^{i_0}_{j[1:i_0]} \oplus \beta) \oplus \bigoplus_{i=i_0+1}^{\ell} c^i_{x_{[1:i]}} \\
&= \bigoplus_{i=1}^{\ell} c^i_{x_{[1:i]}} = [\![\mathbf{r}^{(x)}]\!]_a.
\end{aligned}$$

Case 2: When $j = \alpha_{[1:\ell]}$, we always have $\tilde{c}^1_{x_1} = c^1_{x_1} \oplus \beta$.
For all $1 < i \leq \ell$, since $x_{[1:i]}$ is a prefix of α, we have $\tilde{c}^i_{x_{[1:i]}} = c^i_{x_{[1:i]}}$. Overall, we have $[\![\mathbf{r}^{(x)}]\!]_a \oplus [\![\mathbf{r}^{(x)}]\!]_b = \beta$ since:

$$[\![\mathbf{r}^{(x)}]\!]_b = \tilde{c}^1_{x_1} \oplus \bigoplus_{i=2}^{\ell} \tilde{c}^i_{x_{[1:i]}} = (c^1_{x_1} \oplus \beta) \oplus \bigoplus_{i=2}^{\ell} c^i_{x_{[1:i]}} = [\![\mathbf{r}^{(x)}]\!]_a \oplus \beta.$$

Case 3: When $j > \alpha_{[1:\ell]}$, let i_0 be the first index such that $x_{i_0} > \alpha_{i_0}$:

- If $i_0 = 1$, it means that $x_1 = 1$ and $\alpha_1 = 0$, and we have $\tilde{c}^1_0 = c^1_0 \oplus \beta$.
  For all $1 < i \leq \ell$, $x_{[1:i]}$ is not a prefix of α, hence $\tilde{c}^i_{x_{[1:i]}} = c^i_{x_{[1:i]}}$. In conclusion, we have $[\![\mathbf{r}^{(x)}]\!]_a \oplus [\![\mathbf{r}^{(x)}]\!]_b = \beta$ since:

$$[\![\mathbf{r}^{(x)}]\!]_b = \tilde{c}^1_0 \oplus \bigoplus_{i=2}^{\ell} \tilde{c}^i_{x_{[1:i]}} = (c^1_0 \oplus \beta) \oplus \bigoplus_{i=2}^{\ell} c^i_{x_{[1:i]}} = [\![\mathbf{r}^{(x)}]\!]_a \oplus \beta.$$

- If $i_0 > 1$, we have $x_1 = \alpha_1$, hence $\tilde{c}^1_{x_1} = c^1_{x_1} \oplus \beta$.
  For all $1 < i < i_0$, since $x_{[1:i]}$ is a prefix of α, we have $\tilde{c}^i_{x_{[1:i]}} = c^i_{x_{[1:i]}}$. For all $i > i_0$, since $x_{[1:i-1]}$ is not a prefix of α, we also have $\tilde{c}^i_{x_{[1:i]}} = c^i_{x_{[1:i]}}$. The remaining index to analyse is i_0, where $x_{i_0} = 0$ and $\alpha_{i_0} = 1$, hence $\tilde{c}^{i_0}_j = c^{i_0}_j$. In conclusion, we have $[\![\mathbf{r}^{(x)}]\!]_a \oplus [\![\mathbf{r}^{(x)}]\!]_b = \beta$ since:

$$\begin{aligned}
[\![\mathbf{r}^{(x)}]\!]_b &= \tilde{c}^1_0 \oplus \bigoplus_{i=2}^{i_0} \tilde{c}^i_{x_{[1:i]}} \oplus \tilde{c}^{i_0}_{j[1:i_0]} \oplus \bigoplus_{i=i_0+1}^{\ell} \tilde{c}^i_{x_{[1:i]}} \\
&= (c^1_0 \oplus \beta) \oplus \bigoplus_{i=2}^{i} c^i_{x_{[1:i]}} \oplus c^{i_0}_{j[1:i_0]} \oplus \bigoplus_{i=i_0+1}^{\ell} c^i_{x_{[1:i]}} \\
&= \bigoplus_{i=1}^{\ell} c^i_{x_{[1:i]}} \oplus \beta = [\![\mathbf{r}^{(x)}]\!]_a \oplus \beta.
\end{aligned}$$

□

Lemma 2 (Pseudorandom Output)**.** *When both parties are honest, given $G : \{0,1\}^\kappa \to \{0,1\}^{4\kappa}$ is a secure PRG, the protocol Π_{iDCF} in Figure 5 outputs pseudorandom shares: for each $x \in \bigcup_{\ell=1}^{u}\{0,1\}^\ell$, the output shares $[\![\mathbf{r}^{(x)}]\!]_a$ and $[\![\mathbf{r}^{(x)}]\!]_b$ are computationally indistinguishable from random.*

Proof. Following the correctness of Π_{iDCF} proved in Lemma 1, it remains to show that Bob's output is computationally indistinguishable from a uniformly random share. We follow a GGM tree-style hybrid argument. We define the hybrids as follows:

For each $i \in [u]$ and $j \in [0, 2^i)$, Hybrid (i, j) is defined such that every base values $g^{i}_{j'}$ and payloads $c^{i}_{j'}$ in which $i < i$ or $i = i \wedge j' < 2j$ are sampled uniformly random. The other base values $g^{i}_{j'}$ and payloads $c^{i}_{j'}$ are generated with PRG following Π_{iDCF}. In this construction, Hybrid $(1, 0)$ is the same as the honest execution, where only the key $k = g^0_0$ is sampled uniformly. Meanwhile, in Hybrid $(u, 2^{u-1})$, the shares $[\![\mathbf{r}^{(x)}]\!]_b$ are all uniformly random since all c^u_j are uniformly random.

It remains is to prove that for any $i \in [u]$ and $j \in [0, 2^i)$, Hybrid (i, j) is indistinguishable from Hybrid $(i, j + 1)$. We also note that Hybrid $(i, 2^{i-1})$ and Hybrid $(i + 1, 0)$ are the same, so we can conclude that Hybrid $(1, 0)$ and Hybrid $(u, 2^{u-1})$ are indistinguishable afterward. The indistinguishability between Hybrid (i, j) and Hybrid $(i, j+1)$ relies on the security of PRG G. In particular, We construct an adversary $\mathcal{A}$ that breaks the PRG security game with G as follow:

1. The challenger samples $\eta \leftarrow \{0, 1\}$. If $\eta = 0$, the challenger samples uniformly random $x_1, x_2, x_3, x_4 \in \{0,1\}^\kappa$. Otherwise, the challenger samples uniformly random $x \in \{0,1\}^\kappa$ and computes $(x_1, x_2, x_3, x_4) = G(x)$.
2. $\mathcal{A}$ uniformly samples the base values $g^{i}_{j'}$ and the payloads $c^{i}_{j'}$, when either $i < i$ or $i = i \wedge j' < 2j$. Note that, looking forward, $\mathcal{A}$ will not use g^{i-1}_j when computing the output share to send to Bob. $\mathcal{A}$ now receives x_1, x_2, x_3, x_4 from the challenger and sets:

   $$(g^i_{2j}, g^i_{2j+1}, c^i_{2j}, c^i_{2j+1}) = (x_1, x_2, x_3, x_4).$$

   For other base values $g^{i}_{j'}$ and payloads $c^{i}_{j'}$, $\mathcal{A}$ computes them honestly as stated in the protocol Π_{iDCF}.
3. $\mathcal{A}$ computes all $[\![\mathbf{r}^{(x)}]\!]_a$ honestly and sends these shares to Bob.
4. $\mathcal{A}$ outputs the same output as Bob to the challenger.

□

We now move on to provide a simulation-based security proof for Π_{iDCF} when Alice or Bob is corrupted. We argue security by constructing a PPT simulator Sim that can simulate the view of the corrupted party in both cases.

Alice is corrupted. We construct Sim that takes α from Alice and generates $[\![\mathbf{r}]\!]_a$ to send to $\mathcal{F}_{\mathsf{dFSS}}$. The goal is to simulate (through a sequence of hybrids) Alice's view from the honest protocol execution independent of Bob's input.

Hybrid 0. This hybrid acts as the honest execution of the protocol, where Alice' view is generated with Bob's private inputs β.

Hybrid 1. In this hybrid, we replace Bob's output with $[\![\mathbf{r}^{(x)}]\!]_b = [\![\mathbf{r}^{(x)}]\!]_a \oplus \mathsf{iDCF}_{\alpha,\beta}(x)$ for all $x \in \bigcup_{\ell=1}^{u}\{0,1\}^\ell$. This is identical to the previous hybrid by Lemma 1.

Hybrid 2. For each instance i of $\mathcal{F}_{\mathsf{OT}}$, where $i \in [u]$, we replace Bob's input $m^i_{\overline{\alpha_i}}$ into $\perp$. This is identical to the previous hybrid in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model.

Hybrid 3. Same as the previous hybrid except for step 1(b)i, $i = 1$. We modify the computation of base values and payloads for $(g_0^1, g_1^1, c_0^1, c_1^1)$ into sampled uniformly from $\{0,1\}^\kappa$, instead of being outputs of the PRG, $G(g_0^0)$. This is computationally indistinguishable from the previous hybrid; otherwise, an adversary could break the PRG security, which is shown in more detail in Lemma 2.

For each $j \in [4, u+2]$, Hybrid j is defined as below:

Hybrid j. Same as Hybrid $(j-1)$ except for step 1(b)i, $i = j-2$. We modify the computation of base values and payloads for $(g^i_{\alpha_{[1:i-1]}\|0}, g^i_{\alpha_{[1:i-1]}\|1}, c^i_{\alpha_{[1:i-1]}\|0}, c^i_{\alpha_{[1:i-1]}\|1})$ into sampled uniformly from $\{0,1\}^\kappa$, instead of being outputs of the PRG, $G(g^{i-1}_{\alpha_{[1:i-1]}})$. This is computationally indistinguishable from Hybrid $(j-1)$, via a reduction to the security of PRG. Note that we rely on the fact that $g^{i-1}_{\alpha_{[1:i-1]}}$ is hidden to Alice, which is ensured by OT, where the OT message that contains $g^{i-1}_{\alpha_{[1:i-1]}}$ has been replaced by $\perp$ in Hybrid 2.

Hybrid $(u+3)$. Note we arrive at a hybrid where for any prefix j of α, $|j| = i < u$, we have $g^{i+1}_{j\|0}, g^{i+1}_{j\|1}, c^{i+1}_{j\|0}, c^{i+1}_{j\|1}$ being sampled uniformly from $\{0,1\}^\kappa$.

We now analyze the OT messages. For level $i = 1$, if $\alpha_1 = 0$, we have the message received by Alice in this hybrid being:

$$\begin{aligned} m_0^1 &= (\mathbf{g}_1^1, \quad \mathbf{c}_0^1 \oplus \beta, \quad \mathbf{c}_1^1 \oplus \beta), \\ &= (g_1^1, \quad c_0^1 \oplus \beta, \quad c_1^i \oplus \beta). \end{aligned}$$

We now sample uniformly $(g_1, c_{1,0}, c_{1,1}) \leftarrow\$ \{0,1\}^\kappa$ and change the message m_0^1 into:

$$m_0^1 = (g_0, \quad c_{1,0}, \quad c_{1,1}),$$

which is statistically identical from the previous distribution. We do similarly for the case when $\alpha_1 = 1$, by sampling uniformly $(g_1, c_{i,0}, c_{i,1}) \leftarrow\$ \{0,1\}^\kappa$ and change the message into:

$$m_1^1 = (g_1, \quad c_{1,0}, \quad c_{1,1}).$$

For each level i from 2 to u, if $\alpha_i = 0$, we have the message received by Alice in this hybrid being:

$$\begin{aligned} m_0^i &= \mathbf{g}_1^i, \mathbf{c}_0^i \oplus \beta, \mathbf{c}_1^i \oplus \beta \\ &= \sum_{\substack{j \in [0,2^{i-1}), \\ j \neq \alpha_{[1:i-1]}}} g^i_{j\|1} \oplus g^i_{\alpha_{[1:i-1]}\|1}, \sum_{\substack{j \in [0,2^{i-1}), \\ j \neq \alpha_{[1:i-1]}}} c^i_{j\|0} \oplus c^i_{\alpha_{[1:i-1]}\|0}, \sum_{\substack{j \in [0,2^{i-1}), \\ j \neq \alpha_{[1:i-1]}}} c^i_{j\|0} \oplus c^i_{\alpha_{[1:i-1]}\|1}. \end{aligned}$$

For ease of streamlining, we will still sample uniformly $(g_i, c_{i,0}, c_{i,1}) \leftarrow\$ \{0,1\}^\kappa$ and change the message sent to Alice into:

$$m_0^i = \sum_{\substack{j \in [0,2^{i-1}), \\ j \neq \alpha_{[1:i-1]}}} g^i_{j\|1} \oplus g_i, \sum_{\substack{j \in [0,2^{i-1}), \\ j \neq \alpha_{[1:i-1]}}} c^i_{j\|0} \oplus c_{i,0}, \sum_{\substack{j \in [0,2^{i-1}), \\ j \neq \alpha_{[1:i-1]}}} c^i_{j\|1} \oplus c_{i,1}.$$

If $\alpha_i = 1$, also following a statistical argument, we sample uniformly $(g_i, c_{i,0}, c_{i,1}) \leftarrow\$ \{0,1\}^\kappa$ and change the message sent to Alice into:

$$m_1^i = \sum_{\substack{j \in [0,2^{i-1}), \\ j \neq \alpha_{[1:i-1]}}} g^i_{j\|1} \oplus g_i, \sum_{\substack{j \in [0,2^{i-1}), \\ j \neq \alpha_{[1:i-1]}}} c^i_{j\|0} \oplus c_{i,0}, \sum_{\substack{j \in [0,2^{i-1}), \\ j \neq \alpha_{[1:i-1]}}} c^i_{j\|1} \oplus c_{i,1},$$

which is statistically identical to Hybrid $(u+2)$ since both $c^i_{\alpha_{[1:i-1]}\|0} \oplus \beta$ and $c_{i,0}$ are uniformly random.

Simulator. For all $j \in [0, 2^i)$ such that $j \neq \alpha_{[1:i]}$, the base values and the payloads $(g^i_{j\|0}, g^i_{j\|1}, c^i_{j\|0}, c^i_{j\|1})$ can be computed from $(g^1_0, g^1_1, c^1_0, c^1_0)$, which are sampled uniformly at random. Therefore, Hybrid $(u+3)$ is a valid simulator without knowing Bob's inputs.

Bob is corrupted. Since Bob does not receive any message from the protocol, the simulation is straightforward. The simulator simply follows the protocol honestly to generate Bob's view and sends the resulting shares $\left\{[\![\mathbf{r}^{(x)}]\!]_b\right\}_{x \in \bigcup_{\ell=1}^{u}\{0,1\}^\ell}$ to the ideal functionality. □

C Additional Parameters in Implementation

In this section, we include the additional parameters in our implementation. Table 5 provides the LPN parameters for the VOLE construction [WYKW21].

Parameter Set	n_{extend}	k_{extend}	t_{extend}	n_{pre}	k_{pre}	t_{pre}	n_{base}	k_{base}	t_{base}
2^{12} VOLE extensions	5600	600	700	1400	300	350	800	250	200
2^{16} VOLE extensions	76800	5000	1200	6400	650	800	1500	300	375
2^{20} VOLE extensions	1088000	30500	8500	40000	3000	1250	4400	600	550

Table 5: LPN parameters in the VOLE construction [WYKW21].

Table 6 lists our chosen set of prefix lengths $\mathcal{L}$ used for load balancing between Alice and Bob (Section 7.2). Note that 2δ is the size of each dimension of the mini-universe, and $\log(2\delta)$ is the number of bits used to represent a point in each dimension.

Radius	$\log(2\delta)$	$\mathcal{L}$
10	6	$(2, 4, 6)$
30	7	$(1, 3, 5, 7)$
60	8	$(2, 4, 6, 8)$
120	9	$(1, 3, 5, 7, 9)$
250	10	$(2, 4, 6, 8, 10)$

Table 6: Set of prefix lengths that Bob computes hashes on.