Fuzzy PSI via Oblivious Protocol Routing[⋆]

David Richardson, Mike Rosulek, and Jiayu Xu

Oregon State University, {richdavi,rosulekm,xujiay}@oregonstate.edu

Abstract. In private set intersection (PSI), two parties who each hold sets of items can learn their intersection without revealing anything about their other items. Fuzzy PSI corresponds to a relaxed variant that reveals pairs of items which are "close enough," with respect to some distance metric. In this paper we propose a new protocol framework for fuzzy PSI, compatible with arbitrary distance metrics. We then show how to efficiently instantiate our framework for ℓ_1, ℓ_2, and ℓ_∞ metrics, in a way that uses exclusively cheap symmetric-key operations. One notable feature of our protocol is that it has only logarithmic dependency on the distance threshold, whereas most other protocols have linear (or higher) dependency. For many reasonable combinations of parameters, our protocol has the lowest communication cost of existing fuzzy PSI protocols.

1 Introduction

Private set intersection (PSI) is a special case of secure 2-party computation where the function is the intersection function. In other words, each party has a set of items, and the result of the computation is to reveal the intersection of those sets, without revealing anything else about the private inputs. Since its introduction by Freedman, Nissim, and Pinkas in 2004 [FNP04], PSI has been the focus of significant optimization. Today, PSI is truly practical for real-world privacy-preserving computations. The protocol and efficient implementation of Raghuraman & Rindal [RR22] can compute the intersection of two sets of 1M items each in less than a half second.

Some applications of PSI are hindered by the fact that the data is inherently *noisy.* For example, a single person's biometric information is not measured precisely the same way every time. GPS coordinates, even of a fixed location, are measured with an inherent error. What is needed for examples like these is **fuzzy PSI.**

If Alice has input set A and Bob has input set B, then fuzzy PSI allows them to learn about the pairs $(a, b) \in A \times B$ where a and b are not necessarily identical, but are merely *close.* More precisely, a and b should satisfy $d(a, b) \leq \delta$, where d is a distance metric and δ a publicly agreed similarity threshold.

[⋆] First two authors partially supported by NSF award S2356A

Importance of Euclidean metric in ML. One common source of "noisy" data is complex, high-dimensional, unstructured data. The case of human faces provides a good example. ML autoencoders can encode photos of faces into vectors, such that different images of the same person's face have similar encodings. Fuzzy PSI is a good fit for this kind of data.

The most common way to measure similarity in an autoencoding is the *cosine similarity* measure — i.e., the cosine of the angle between the two vectors. Similar vectors have cosine very close to 1.

For our purposes, it is important to understand that cosine similarity can be expressed in terms of Euclidean distance:

Proposition 1. *If all datapoints x, y are unit vectors, then $\cos(x, y) \geq 1 - \delta$ if and only if $d(x, y) \leq \delta'$, where d is the Euclidean (ℓ_2) distance metric, and δ' is a function of δ alone.*

Thus, for applications of fuzzy PSI to ML-driven, high-dimensional, unstructured data sets, Euclidean (ℓ_2) distance is the most important distance metric.

Symmetric vs asymmetric cryptography. An important qualitative property of practical MPC protocols is whether they rely on symmetric-key or public-key primitives. For example, many PSI protocols (starting with [PSZ14] and including many others [KKRT16,PRTY19,CM20,GRS22]) are based on oblivious transfer (OT) extension [Bea96,IKNP03]. After performing λ (e.g, 128) base OT instances, which require public-key operations, the remainder of the protocol uses exclusively symmetric-key operations. In other words, the marginal cost per item (in a PSI protocol) involves only symmetric-key primitives.

Other PSI approaches require a linear (in the size of the input sets) number of public-key operations. Of course, just because protocol A uses symmetric-key operations and protocol B uses public-key operations, it is no guarantee that protocol A will be faster in practice. However, the incredible advancement in (plain) PSI protocol performance over the last decade would not have been possible without the ability to base them on fast symmetric-key operations.

1.1 State of the art

Structure-aware PSI (saPSI) refers to a special case of PSI where one party's input set has a publicly known structure. The cost of an saPSI protocol should scale not with the cardinality of the structured set, but its description size. saPSI can be used for fuzzy PSI by having one party expand their set of points S to the set of nearby points $S^+ = \{x \mid d(x, S) \leq \delta\}$. The resulting set is much larger, but it is public knowledge that it is structured as the union of metric balls. Thus, an saPSI protocol involving S^+ produces the desired fuzzy PSI functionality. Several papers propose efficient saPSI protocols [GRS22,GRS23,GGM24]. These all are based on symmetric-key techniques, but the underlying results are currently limited to the ℓ_∞ metric and (with an penalty that is exponential in the dimension) ℓ_1 metric.

van Baarsen and Pu [vP24] propose a fuzzy PSI protocol for ℓ_p metrics. It uses an additively homomorphic encryption scheme to perform comparisons under this metric — i.e., it uses a linear amount of public-key operations. Furthermore, it has (at least) linear dependence on the distance threshold δ. More precisely, when identifying points within distance δ in the ℓ_p metric, the protocol has communication $\Omega(\delta^p)$.

In very recent work, Gao et al. [GQL$^+$24] propose a new fuzzy PSI approach based on a technique that they call **fuzzy mapping.** A common idea in fuzzy PSI protocols (including our own) is to limit the necessary number of comparisons by first mapping items into bins, in a way that respects close pairs of items. In most fuzzy PSI protocols, this mapping process is local/non-interactive, but Gao et al. propose new *interactive* methods of doing this mapping. They provide instantiations for ℓ_∞, ℓ_p, and Hamming metrics. Both their mapping process and their subsequent distance comparison subprotocols use (public-key) additively homomorphic techniques. Their protocol cost also scales linearly with the distance threshold δ.

1.2 Our results & technical overview

Our main result is a new, efficient fuzzy PSI protocol framework, which can be instantiated for a variety distance metrics.

Starting point: Our starting point is the PSI technique of Cho, Dachman-Soled, and Jarecki [CDJ16], hereafter CDJ. Their protocol compiles a private equality test (PEQT) protocol into a PSI protocol. A PEQT protocol takes an input x from Alice and input y from Bob, and reveals (to one of the parties) whether $x = y$, and nothing else about the inputs. One might interpret a PEQT protocol as a PSI protocol for singleton sets. In the CDJ protocol, each party runs one PEQT protocol for each of its items, using that item as input. When Alice and Bob have a common item x, they each have a PEQT instance associated with that item, and they would like these instances to "talk to each other." The challenge is that this must happen while still hiding the identity of each party's items.

The CDJ approach is for both parties to encode their PET protocol messages in a polynomial. More precisely, Alice interpolates a polynomial P such that $P(x)$ equals the next protocol message in her PET protocol instance associated with x, and sends P to Bob. For each of Bob's inputs y, he interprets $P(y)$ as a protocol message in his PET protocol instance associated with y. Then, similar to Alice, he encodes the set of PET responses into another polynomial Q, which he sends to Alice.

Of course, if Alice and Bob have a common item x, then their corresponding PET instances will be communicating on the "same frequency" of the polynomials, and the PET instances will indicate a match. Furthermore, if the PET protocol messages have a certain pseudorandomness property, then it can be shown that the polynomials P and Q leak nothing about a party's inputs.

Our modifications: We generalize the CDJ approach to the fuzzy PSI setting in the following ways.

First, the parties will hash their items into *bins* according to some abstract hashing scheme. Each item may be hashed into several bins, and the universe of possible bins may be exponentially large. The hashing scheme should have the property (which we call *conditionally overlapping*) that if Alice and Bob hold x and y which are "close enough," then some bin will contain both x and y.

For the sake of simplicity, suppose that each bin is guaranteed to have at most one item per party. (Our general framework handles the more general case, where bins may have many items.) For each non-empty bin, the parties run an instance of a *private proximity protocol,* which takes private inputs x and y, and outputs a bit indicating whether x and y are closer than a public threshold. The parties encode their protocol messages into a polynomial P such that $P(\beta)$ is the next protocol message for the item in bin β. (Polynomial interpolation is just one way of encoding a key-value map; our protocol is written abstractly in terms of an *oblivious key-value store.*)

The correctness and privacy analysis of our protocol is similar to that of CDJ. In fact, the CDJ protocol is obtained as a special case of our framework, where the hashing function is the identity function and the proximity protocol is an equality protocol. We present our protocol as a general framework that combines any conditionally overlapping hashing schemes and any proximity-subprotocol functionality. We prove security against semi-honest adversaries.

Instantiating our framework. For ℓ_p and ℓ_∞ metrics, garbled circuits are the most straightforward way to instantiate the proximity protocol. The result is concretely efficient, with relatively low communciation.

Given two inputs, the garbled circuit can compute their distance and compare it to a fixed threshold. We find that the arithmetic garbling scheme of [BMR16] is more efficient than standard boolean garbling for ℓ_2 comparisons, because it supports extremely efficient squaring as well as allowing the garbler to easily hard-code their point into the circuit. On the other hand, standard boolean garbling is more efficient for ℓ_1 and ℓ_∞ metrics.

Efficiency. Among existing fuzzy PSI protocols, ours has the lowest communication when the threshold δ is moderately large (e.g., around and above 30). In 2 dimensions, with $\delta = 30$ and 2^{16} items, our protocol requires 455MB of communication for ℓ_1 distance, 1.1GB for ℓ_2 distance, and 453MB for ℓ_∞ distance. These represent an improvement of 4.8×, 3.3×, and 4.7× over the next closest competitor.

2 Preliminaries

We use "$\equiv$" to denote that two distributions are identical, and use "$\approx$" to denote that they are indistinguishable. For a functionality $\mathcal{F}$, we use $\mathcal{F}_1(A, B)$ to indicate the output of party 1 from an execution of $\mathcal{F}$ where party 1's input is A and party 2's input is B.

2.1 Oblivious Key-Value Stores

Given a set of distinct keys $\{k_1, \ldots, k_n\}$, if one interpolates a polynomial P through the points (k_i, v_i), and the v_i values are chosen uniformly, then the coefficients of P perfectly hide the k_i values. Oblivious key-value stores (OKVS) were introduced by Garimella et al. [GPR$^+$21] as a generalization of this useful property of polynomial interpolation.

Definition 1. *For key space $\mathcal{K}$ and value space $\mathcal{V}$, a **key-value store** (KVS) is defined by a pair of algorithms:*

- $\mathsf{Encode}(s \subseteq \mathcal{K} \times \mathcal{V})$ *outputs an object S or an error symbol $\perp$.*
- $\mathsf{Decode}(S, k \in \mathcal{K})$ *outputs a value $v \in \mathcal{V}$.*

A KVS is *correct* if for every key-value set s with distinct keys and every pair $(k, v) \in s$, $\mathsf{Decode}(\mathsf{Encode}(s), k) = v$ with overwhelming probability. Further, it is *oblivious* if the following two oracles are indistinguishable:

```
OKVS-LEFT($K_1, K_2$):
    assert $|K_1| = |K_2|$
    $V \leftarrow \mathcal{V}^{|K_1|}$
    $s = \{(K_1[i], V[i])\}_{i=1}^{|K_1|}$
    Output $\mathsf{Encode}(s)$
```

$\approx$

```
OKVS-RIGHT($K_1, K_2$):
    assert $|K_1| = |K_2|$
    $V \leftarrow \mathcal{V}^{|K_2|}$
    $s = \{(K_2[i], V[i])\}_{i=1}^{|K_2|}$
    Output $\mathsf{Encode}(s)$
```

The previous property says, roughly: if one encodes an OKVS with *entirely random* values, then the OKVS output hides the choice of keys. An even more general property holds as well: OKVS output leaks only the identity of keys whose corresponding values are *not* chosen randomly. More formally, the following two oracles are indistinguishable. K_1, V_1 denote the non-random keys/values, and the adversary is unable to distinguish whether random values were associated with keys K_2 or K_3:

```
OKVS-PARTIAL-LEFT($K_1, K_2, K_3, V_1$):
    assert $|K_2| = |K_3|$ and $|K_1| = |V_1|$
        and $K_1 \cap K_2 = \emptyset$ and $K_1 \cap K_3 = \emptyset$
    $V_2 \leftarrow \mathcal{V}^{|K_2|}$
    $s = \{(K_1[i], V_1[i])\}_{i=1}^{|K_1|} \cup \{(K_2[j], V_2[j])\}_{i=1}^{|K_2|}$
    Output $\mathsf{Encode}(s)$
```

$\approx$

```
OKVS-PARTIAL-RIGHT($K_1, K_2, K_3, V_1$):
    assert $|K_2| = |K_3|$ and $|K_1| = |V_1|$
        and $K_1 \cap K_2 = \emptyset$ and $K_1 \cap K_3 = \emptyset$
    $V_3 \leftarrow \mathcal{V}^{|K_3|}$
    $s = \{(K_1[i], V_1[i])\}_{i=1}^{|K_1|} \cup \{(K_3[j], V_3[j])\}_{i=1}^{|K_3|}$
    Output $\mathsf{Encode}(s)$
```

The previous definition is obtained as the special case where $K_1 = \emptyset$.

Expand-then-encode. We overload the notation of Encode in the following way: $\mathsf{Encode}(s, h)$ means: add dummy key-value pairs to s until $|s| = h$, and then encode as usual. The dummy values are chosen uniformly, so the choice of dummy keys is irrelevant.

```
Encode($s, h$):
    while $|s| < h$:
        $k \leftarrow \{l \in \mathcal{K} \mid \neg\exists v : (l, v) \in s\}$
        $v \leftarrow \mathcal{V}$
        add $(k, v)$ to $s$
    Output Encode($s$)
```

Thus, the following two oracles are indistinguishable (note that it is no longer necessary that $|K_1| = |K_2|$):

```
OKVS-LEFT($K_1, K_2, h$):
    assert $|K_1| \le h$ and $|K_2| \le h$
    $V \leftarrow \mathcal{V}^{|K_1|}$
    $s = \{(K_1[i], V[i])\}_{i=1}^{|K_1|}$
    Output Encode($s, h$)
```

$\approx$

```
OKVS-RIGHT($K_1, K_2, h$):
    assert $|K_1| \le h$ and $|K_2| \le h$
    $V \leftarrow \mathcal{V}^{|K_2|}$
    $s = \{(K_2[i], V[i])\}_{i=1}^{|K_2|}$
    Output Encode($s, h$)
```

Instantiations. The simplest (and size-optimal) OKVS is based on polynomials: Encode is polynomial interpolation, and Decode is polynomial evaluation. However, the cost of encoding n key-value pairs is $O(n \log^2 n)$. Other OKVS with linear-time encoding are proposed in [GPR+21].

2.2 Conditionally Overlapping Hash Pairs

In our protocol, each party uses a hashing function to assign each of their items to a set of *bins*. The hashing method must have the property that "similar items" must be assigned to a common bin.

Definition 2. *Let $\mathcal{L}$ be a set of items, $\mathcal{K}$ be a set of bin identifiers, and f be a symmetric binary function $f : \mathcal{L} \times \mathcal{L} \to \{0, 1\}$. A* ***conditionally overlapping hash pair*** *for f is a pair of hash functions $\mathcal{H}_1, \mathcal{H}_2 : \mathcal{L} \to \mathsf{PowerSet}(\mathcal{K})$, along with positive integer constants h_1 and h_2, such that for any $i, j \in \mathcal{L}$, $|\mathcal{H}_1(i)| \le h_1$, $|\mathcal{H}_2(j)| \le h_2$, and $f(i, j) = 1 \implies \mathcal{H}_1(i) \cap \mathcal{H}_2(j) \ne \emptyset$.*

Each hash function assigns item x to the bins $\mathcal{H}(x)$. For a set S of inputs, we use $\mathcal{H}(S)$ to denote the bins that are assigned at least one item from S — i.e., $\mathcal{H}(S) = \{\beta \mid \exists\, s \in S : \beta \in \mathcal{H}(s)\}$ (where $\mathcal{H}$ is either $\mathcal{H}_1$ or $\mathcal{H}_2$). When S is the input of a certain party, we will sometimes refer to the bins in $\mathcal{H}(S)$ as the **active bins** of that party. We write $\mathcal{H}^{-1}(\beta, S)$ to indicate the set of items in S that hash to β, i.e., $\mathcal{H}^{-1}(\beta, S) = \{s \in S \mid \beta \in \mathcal{H}(S)\}$. Finally, we sometimes refer to the binary function f as a **similarity function**.

The above definition assumes the hash functions to be stateless — i.e., the assignment of item x to bins does not depend on other items being hashed. This is merely to simplify the notation surrounding hashing. However, one could also consider methods of assigning items to bins, based on a *global* view of all the items being hashed; our general results would still hold.

Finally, we sometimes use $\mathcal{F}_1^\beta(A,B)$ as a shorthand for $\mathcal{F}_1\Big(\mathcal{H}_1^{-1}(\beta,A),\ \mathcal{H}_2^{-1}(\beta,B)\Big)$. Intuitively, $\mathcal{F}_1^\beta(A,B)$ is the output of party 1 from $\mathcal{F}$ in which party 1's input is the subset of points in A that hash to β and party 2's input is the subset of points in B that hash to β.[1]

2.3 Subprotocols

Our main construction encodes the protocol messages of certain subprotocols into an OKVS. Thus, these protocol messages need to enjoy certain pseudorandomness properties. In this section we discuss the necessary properties of these subprotocol in an abstract manner, without specifying their functionality; i.e., we consider subprotocols that realize some unspecified deterministic two-party functionality $\mathcal{F}$. For simplicity, we consider 1-round subprotocols, although our results would generalize in a natural way to many-round subprotocols.

Subprotocol Interface We refer to a one-round protocol as a *subprotocol* if, for input space $\mathcal{I}$ and message spaces $\mathcal{M}_1, \mathcal{M}_2$, it can be expressed as algorithms with the following behavior:

- $\mathsf{PROT}_1(a \in \mathcal{I})$ outputs (M_1, σ) where $M_1 \in \mathcal{M}_1$;
- $\mathsf{PROT}_2(M_1 \in \mathcal{M}_1, b \in \mathcal{I})$ outputs $M_2 \in \mathcal{M}_2$;
- $\mathsf{PROT}_3(\sigma, M_2 \in \mathcal{M}_2)$ outputs party 1's final output.

In a standard execution of a subprotocol, party 1 runs $(M_1, \sigma) \leftarrow \mathsf{PROT}_1(a)$, sends a message M_1 to party 2, and keeps σ as its internal state. Then party 2, upon receiving M_1, runs $M_2 \leftarrow \mathsf{PROT}_2(M_1, b)$ and sends a message M_2 to party 1. Finally, party 1, upon receiving M_2, outputs $\mathsf{PROT}_3(\sigma, M_2)$ (party 2 outputs nothing).

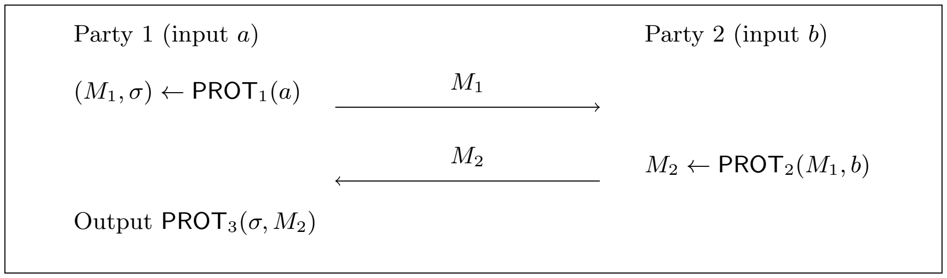
Party 1 (input a)
Party 2 (input b)
$(M_1, \sigma) \leftarrow \mathsf{PROT}_1(a)$
M_1
M_2
$M_2 \leftarrow \mathsf{PROT}_2(M_1, b)$
Output $\mathsf{PROT}_3(\sigma, M_2)$

Fig. 1. Execution of a subprotocol

Note that the interface above implicitly requires the functionality $\mathcal{F}$ to have the same input space for the two parties, and one party should not receive any output.

[1] In our protocols, party 1 exclusively uses $\mathcal{H}_1$ and party 2 exclusively uses $\mathcal{H}_2$, so which hash function is used for this is unambiguous.

Pseudorandomness We require subprotocol messages to appear pseudorandom under various scenarios:

Definition 3. *A subprotocol has a* ***pseudorandom first message****, or* ***first pseudorandomness****, if the following oracles are indistinguishable:*

```
PROT_1-REAL($a \in \mathcal{I}$):
    $(M_1, \sigma) \leftarrow \mathsf{PROT}_1(a)$
    Output $M_1$
```

$$\approx$$

```
PROT_1-RANDOM($a \in \mathcal{I}$):
    $M_1 \leftarrow \mathcal{M}_1$
    Output $M_1$
```

Definition 4. *A subprotocol has a* ***pseudorandom second message****, or* ***second pseudorandomness****, if the second message is indistinguishable from uniform, when responding to a uniformly random first message. That is,*

```
PROT_2-REAL($b \in \mathcal{I}$):
    $M_1 \leftarrow \mathcal{M}_1$
    $M_2 \leftarrow \mathsf{PROT}_2(M_1, b)$
    Output $(M_1, M_2)$
```

$$\approx$$

```
PROT_2-RANDOM($b \in \mathcal{I}$):
    $M_1 \leftarrow \mathcal{M}_1$
    $M_2 \leftarrow \mathcal{M}_2$
    Output $(M_1, M_2)$
```

First and second pseudorandomness combined imply that for any inputs $a, b \in \mathcal{I}$, the joint distribution of the two protocol messages M_1 and M_2 (i.e., from the perspective of an eavesdropper) is indistinguishable from uniformly random. We may use the term *pseudorandomness* to refer to the combined property.

We require a final property. Whenever the protocol inputs are far (with respect to a similarity function f), we require that the second protocol messages is pseudorandom, even from the perspective of an honest party 1.

Definition 5. *For a similarity function f, a subprotocol with input elements in the domain of f has* ***f-disjoint pseudorandomness****, if the second message is indistinguishable from uniformly random, even given a real first message and its internal state, when the two input sets are "disjoint" according to similarity function f. That is,*

```
PROT_2-REAL($a, b \in \mathcal{I}$):
    assert $f(a, b) = 0$
    $(M_1, \sigma) \leftarrow \mathsf{PROT}_1(a)$
    $M_2 \leftarrow \mathsf{PROT}_2(M_1, b)$
    Output $(M_1, \sigma, M_2)$
```

$$\approx$$

```
PROT_2-RANDOM($a, b \in \mathcal{I}$):
    assert $f(a, b) = 0$
    $(M_1, \sigma) \leftarrow \mathsf{PROT}_1(a)$
    $M_2 \leftarrow \mathcal{M}_2$
    Output $(M_1, \sigma, M_2)$
```

(We slightly abuse notations and write $A, B \subseteq \mathcal{I}$ instead of $A, B \in \mathcal{I}$, i.e., $\mathcal{I}$ is the universe of elements that might be in a party's input set, rather than a set of parties' input sets.[2])

[2] We might also allow parties to have some auxiliary input $\mathsf{aux} \in \mathsf{auxspace}$ apart from the sets A, B, which is unrelated to f-disjoint pseudorandomness (and is thus not shown in the definition). We view $\mathcal{I}$ as the input space, rather than $\mathsf{PowerSet}(\mathcal{I}) \times \mathsf{auxspace}$. Looking ahead, in our context the auxiliary input will be a bin.

A distinguisher can run $\mathsf{PROT}_3(\sigma, M_2)$ to check the output, so f-disjoint pseudorandomness implies the following: for any A, $\mathsf{PROT}_3(\sigma, M_2)$ ran with an internal state σ produced by A and a uniformly random M_2, is indistinguishable from $\mathcal{F}_1(A, B)$ for any B that is f-disjoint from A. This in turn means that for any A and any B_1, B_2 both f-disjoint from A, $\mathcal{F}_1(A, B_1)$ and $\mathcal{F}_1(A, B_2)$ are indistinguishable.

3 Main Protocol

We motivate our main protocol with a concrete example. Suppose PROT is a 1-round protocol, that takes as input one item from each party and returns a boolean indicating whether those items are "close" with respect to some similarity function f. You might think of PROT as a fuzzy-PSI protocol for singleton sets. Suppose we also have a conditionally-overlapping hash pair $(\mathcal{H}_1, \mathcal{H}_2)$ such that if $f(a, b) = 1$ then $\mathcal{H}_1(a) \cap \mathcal{H}_2(b) \neq \emptyset$ — i.e., similar items are mapped to a common bin. Then our construction gives a full-fledged protocol for fuzzy PSI on large sets.

We present the main construction as a generic compiler, which securely runs a subprotocol on the contents of each bin and assembles the results — but without revealing the identities of active bins, or the items assigned to the bins. The functionality that we compute is described below:

$\mathcal{F}_{\text{Bins}}$

Parameters:

- hash functions $\mathcal{H}_1, \mathcal{H}_2 : \mathcal{I} \to \mathsf{PowerSet}(\mathsf{Binspace})$;
- a deterministic two-party functionality $\mathcal{F}$, where each party's input is a subset of $\mathcal{I}$.
- upper bounds n_A and n_B on parties' input sets

Behavior:

await input $A \subseteq \mathcal{I}$ from party 1, where $|A| \leq n_A$
await input $B \subseteq \mathcal{I}$ from party 2, where $|B| \leq n_B$
output $\bigcup_{\beta \in \mathsf{Binspace}} \mathcal{F}_1\Big(\mathcal{H}_1^{-1}(\beta, A),\ \mathcal{H}_2^{-1}(\beta, B)\Big) \times \{\beta\}$ to party 1

Examples. Suppose $(\mathcal{H}_1, \mathcal{H}_2)$ are simply the identity maps: $\mathcal{H}_i(x) = \{x\}$. And suppose $\mathcal{F}(a, b)$ returns $\{a\}$ if $a = b$ and $\emptyset$ otherwise. Then $\mathcal{F}_{\text{Bins}}(A, B)$ reveals $A \cap B$ to Alice. This corresponds to the special case of plain PSI, and the protocol is extremely similar to the CDJ protocol [CDJ16].

Suppose $(\mathcal{H}_1, \mathcal{H}_2)$ are conditionally overlapping with respect to a similarity function f, which assign at most one item to each bin. And suppose $\mathcal{F}(a, b)$ returns $\{a\}$ if $f(a, b) = 1$ and $\emptyset$ otherwise. Then $\mathcal{F}_{\text{Bins}}(A, B)$ reveals to Alice information that can be inferred from:

$$\{\beta \mid \exists b \in B, a \in A : f(a,b) = 1 \text{ and } a, b \text{ both assigned to bin } \beta\}.$$

Alice learning the identity of a bin is equivalent to her learning the identity of her item, since each bin contains at most one of her items. She learns which of her items was near to an item in B, but not its identity. In case her items are assigned to several bins, she learns which of those bins contained the close item.

On the other hand, suppose $\mathcal{F}(a,b)$ returns $\{b\}$ if $f(a,b) = 1$ and $\emptyset$ otherwise. Then $\mathcal{F}_{\text{Bins}}(A,B)$ reveals to Alice information that can be inferred from:

$$\{b \in B \mid \exists a \in A : f(a,b) = 1\}.$$

Knowing the identities of items $b \in B$, it is possible for Alice to deduce in which bins a match was found.

3.1 The Construction

Our protocol for the $\mathcal{F}_{\text{Bins}}$ functionality is given in Figure 3.1. It follows the main ideas described in Section 1.2. Namely, the parties hash their items into bins according to a suitable hash pair. They run an instance of a suitable subprotocol for the contents of each bin, and encode the bin-to-protocol-message mapping in an OKVS.

```
Party 1($A \subseteq \mathcal{I}$)                                  Party 2($B \subseteq \mathcal{I}$)
$S := \emptyset$
**for** each $\beta \in \mathcal{H}_1(A)$ :
    $(M[\beta], \sigma[\beta]) \leftarrow \mathsf{PROT}_1(\mathcal{H}_1^{-1}(\beta, A))$
    add $(\beta, M[\beta])$ to $S$
$P := \mathsf{Encode}_1(S, |A| \cdot h_1 + 1)$
                                   $\xrightarrow{P}$
                                                                    $S' := \emptyset$
                                                                    **for** each $\beta' \in \mathcal{H}_2(B)$ :
                                                                        $M_1 := \mathsf{Decode}_1(P, \beta')$
                                                                        $M'[\beta'] \leftarrow \mathsf{PROT}_2(M_1, \mathcal{H}_2^{-1}(\beta', B))$
                                                                        add $(\beta', M'[\beta'])$ to $S'$
                                                                    $Q := \mathsf{Encode}_2(S', |B| \cdot h_2 + 1)$
                                   $\xleftarrow{Q}$
$O := \emptyset$
**for** each $\beta \in \mathcal{H}_1(A)$ :
    $M_2 := \mathsf{Decode}_2(Q, \beta)$
    $o := \mathsf{PROT}_3(\sigma[\beta], M_2)$
    add $o \times \{\beta\}$ to $O$
Output $O$
```

Fig. 2. Main Protocol: Computes $\mathcal{F}_{\text{Bins}}$

Theorem 1. *Let $\mathcal{F}$ be a deterministic two-party functionality as described in the parameters to $\mathcal{F}_{Bins}$. Let f be a similarity function with input space $\mathcal{I}$. Suppose the protocol in section 3.1 uses the following building blocks:*

1. $\mathcal{H}_1, \mathcal{H}_2 : \mathcal{I} \to \mathsf{PowerSet}(\mathsf{Binspace})$ *are a conditionally-overlapping hash pair for f (with size limits h_1, h_2), where* $\mathsf{Binspace}$ *is the space of potential bins.*
2. PROT *is a correct, pseudorandom, and f-disjoint pseudorandom protocol realizing $\mathcal{F}$ in the semi-honest setting, with* $\mathsf{PROT}_1, \mathsf{PROT}_2$ *having messages in $\mathcal{M}_1, \mathcal{M}_2$, respectively.*
3. $(\mathsf{Encode}_1, \mathsf{Decode}_1)$ *and* $(\mathsf{Encode}_2, \mathsf{Decode}_2)$ *are OKVSs with key space* $\mathsf{Binspace}$ *and value spaces $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively.*

Then the protocol realizes $\mathcal{F}_{Bins}$ (with parameters $\mathcal{H}_1$, $\mathcal{H}_2$, and $\mathcal{F}$) in the semi-honest setting.

Functionalities	
$\mathcal{F}_{\text{Bins}}$	The functionality that our protocol realizes. It is a two-party functionality whose inputs are $A, B \subseteq \mathcal{I}$ and outputs the result of the underlying functionality $\mathcal{F}$ run over A and B hashed with $\mathcal{H}_1$ and $\mathcal{H}_2$.
$\mathcal{F}$	An arbitrary deterministic two-party functionality whose inputs are subsets of $\mathcal{I}$.
Building blocks	
$(\mathcal{H}_1, \mathcal{H}_2)$	A conditionally-overlapping hash pair (section 2.2) for similarity function f, consisting of hash functions from $\mathcal{I}$ to $\mathsf{PowerSet}(\mathsf{Binspace})$.
$(\mathsf{Encode}_1, \mathsf{Decode}_1)$, $(\mathsf{Encode}_2, \mathsf{Decode}_2)$	OKVSs (section 2.1) over $(\mathsf{Binspace}, \mathcal{M}_1)$ and $(\mathsf{Binspace}, \mathcal{M}_2)$, respectively.
PROT	A correct and pseudorandom subprotocol (section 2.3) with f-disjoint pseudorandomness that realizes $\mathcal{F}$.
Spaces	
$\mathcal{I}$	– The input space of the protocol. – The input space of the subprotocol (excluding the auxilliary information which is a bin). – The input space of $(\mathcal{H}_1, \mathcal{H}_2)$.
$\mathsf{Binspace}$	– The output space of $(\mathcal{H}_1, \mathcal{H}_2)$. – The key space of both OKVSs.
$\mathcal{M}_1, \mathcal{M}_2$	– The message space of PROT_1 and PROT_2, respectively. – The value space of $(\mathsf{Encode}_1, \mathsf{Decode}_1)$ and $(\mathsf{Encode}_2, \mathsf{Decode}_2)$, respectively.
Protocol parameters (section 3)	
A, B	Party 1 and Party 2's input, respectively; a set of points in $\mathcal{I}$.
a, b	A point in A and B, respectively.
σ	Party 1's internal state.
FPSI parameters (section 4)	
$\mathscr{d}$	The number of dimensions of $\mathcal{I}$.
δ	The distance threshold for d
u	the bit length of integers for the subprotocol.
$(H_n, H_{n+1}) \mid n \in \{1, 3, 5\}$	Conditionally-overlapping hash pairs for ℓ_p; section 4.1.
s	The number of dimensions party 1 "searches over"; section 4.1.
Other parameters	
f	A similarity function (section 2.2) over $\mathcal{I}$.
h_1, h_2	The maximum number of bins an item can hash to for $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively.
β	A bin identifier in $\mathsf{Binspace}$.
Notation	
$\mathcal{H}(S)$	The set of bins that some item in a set S hashes to. Formally, $\mathcal{H}(S) = \{\beta \mid \exists\, s \in S : \beta \in \mathcal{H}(s)\}$
$\mathcal{H}^{-1}(\beta, S)$	The set of items in a set S that hash to β. Formally, $\mathcal{H}^{-1}(\beta, S) = \{s \in S \mid \beta \in \mathcal{H}(S)\}$
$\mathcal{F}_1(A, B)$	The output of party 1 with input A from $\mathcal{F}$ when party 2 has input B.
$\mathcal{F}_1^{\beta}(A, B)$	The output of party 1 from $\mathcal{F}$ in which party 1's input is the subset of points in A that hash to β and party 2's input is the subset of points in B that hash to β. Formally, $\mathcal{F}_1^{\beta}(A, B) = \mathcal{F}_1\Big(\mathcal{H}_1^{-1}(\beta, A),\ \mathcal{H}_2^{-1}(\beta, B)\Big)$

Table 1. Variables

For convenience, we include a glossary of variable names in table 1.

We give an abbreviated proof sketch here; a full proof can be found in appendix A.2.

Proof (sketch). The proof considers three cases: both parties are honest (i.e., correctness), party 1 is corrupted, and party 2 is corrupted.

Correctness. Party 1's output is a set O consisting of the subprotocol outputs for inputs $\mathcal{H}_1^{-1}(\beta, A)$ over all $\beta \in \mathcal{H}_1(A)$. If β is an active bin for both parties, then the subprotocol is run on "real" inputs, so by correctness of the subprotocol (and correctness of the OKVS), the output is $\mathcal{F}_1^b(A, B) \times \{\beta\}$ with overwhelming probability. Otherwise β is an active bin for party 1 but not party 2; in this case the two parties' inputs in the subprotocol, $\mathcal{H}_1^{-1}(\beta, A)$ and $\mathcal{H}_2^{-1}(\beta, B)$, are f-disjoint, and Q encodes some random-looking values. By f-disjoint pseudorandomness of the subprotocol, party 1's behavior is indistinguishable from a real execution, so its output is again $\mathcal{F}_1^b(A, B) \times \{\beta\}$ with overwhelming probability.

Corrupt party 1.

```
$\underline{\textsc{sim}(A, \mathcal{F}_1(A, B), |B|):}$

  for each $\beta_I \in \mathcal{H}_1(A)$:
    if $\mathcal{F}_1^{\beta_I}(A, B) \neq \emptyset$:
      $(M[\beta_I], M'[\beta_I], \sigma[\beta_I]) \leftarrow \mathsf{SIM}_1(\mathcal{H}_1^{-1}(\beta_I, A), \mathcal{F}_1^{\beta_I}(A, B))$
      add $(\beta_I, M'[\beta_I])$ to $S'$
    else:
      $(M[\beta_I], \sigma[\beta_I]) \leftarrow \mathsf{PROT}_1(\mathcal{H}_1^{-1}(\beta_I, A))$

  $Q := \mathsf{Encode}(S', |B| \cdot h_2 + 1)$

  output $(Q, \sigma)$
```

The simulator is shown above; we briefly argue for indistinguishability of party 1's views between the real world and the ideal world. Party 1's view consists of Q and a list of internal states σ, which contains the randomness of party 1 in the subprotocol. We gradually move from the real world to the ideal world, changing the game's behavior on bins from real to simulated one by one. More concretely, let $(\beta_1, ..., \beta_M)$ be an arbitrary ordering of the bins in $\mathcal{H}_1(A) \cup \mathcal{H}_2(B)$. We consider a series of hybrids indexed from 0 to M: in hybrid t, the behavior of the subprotocol on the first t bins is simulated and the behavior for the remaining bins is real (i.e., unchanged). As a result, hybrid 0 is identical to the real view, and hybrid M is identical to the ideal view. Then we only need to show that hybrid t and hybrid $t + 1$ are indistinguishable.

For a particular bin β, there are four cases to consider:

1. If a bin β is an active bin of party 2 but not party 1, second pseudorandomness of the subprotocol implies that party 2's subprotocol message for that

bin is pseudorandom. This makes the point encoded over β indistinguishable from a random point, so the simulator can ignore β and encode a random point instead.
2. If β is an active bin of both parties, but $\mathcal{F}_1^b(A, B) = \emptyset$, f-disjoint pseudorandomness implies that party 2's subprotocol message for that bin is pseudorandom. As above, the simulator can encode a random point. However, it does need to run PROT_1 to generate an internal state σ.
3. If β is an active bin of both parties, and $\mathcal{F}_1^b(A, B) \neq \emptyset$, the simulator can learn $\mathcal{F}_1^b(A, B)$ from $\mathcal{F}_1(A, B)$, and then use the subprotocol simulator for corrupt party 1 to simulate party 2's subprotocol message. This simulator also provides the internal state.
4. If β is an active bin of party 1 but not party 2, Q is not encoded over β. Because this bin does not affect Q, all that needs to be done is run PROT_1 to generate an internal state σ.

A particular intricacy of these cases is that the simulator cannot differentiate between the conditions in cases 2 and 4; however, this does not pose a problem for us, as the simulator's behavior is the same in these cases — in both cases, all the simulator does is run PROT_1 to generate an internal state.

Corrupt party 2. Party 2's view only includes P and a random tape. Because the adversary does not see party 1's randomness, first pseudorandomness of the protocol implies that party 1's subprotocol messages are pseudorandom, so the simulator can simply encode dummy points to simulate P.

4 Instantiating our Framework for Fuzzy PSI

In this section we describe how to instantiate our general protocol paradigm to achieve a fuzzy PSI protocol for L_1, L_2, and L_∞ distances. We first describe hashing methods, and then describe a proximity subprotocol for Euclidean distances.

4.1 Hashing for Minkowski Distance

In this section, we introduce conditionally-overlapping hash pairs for Minkowski (ℓ_p) distances. Our hashing scheme is based on the idea of placing each point into "bins", which form a grid over the input space.

Definition 6. *Let $\mathbb{Z}_m$ be the ring of integers mod m for some positive integer m and $\mathscr{d}, \delta$ be positive integers. Define the bin of a point $i \in \mathbb{Z}_m^{\mathscr{d}}$ as*

$$B(i) = (\lfloor i[j]/2\delta \rfloor)_{j=1}^{\mathscr{d}}$$

The bins are $\mathscr{d}$-dimensional hypercubes with sides of length 2δ. This specific length is the smallest that ensures that a point can only be "close" to two bins in any one dimension. Each bin is represented by a point in $\mathbb{Z}_m^{\mathscr{d}}$, which is the bin identifier. Roughly, the point representing a bin is the "location" of that bin, divided by 2δ on each dimension.

Next, we present locality-sensitive hashes based on bins.

Definition 7. *Let d be the distance function for an arbitrary Minkowski distance ℓ_p, where p is a positive integer. Define $H_1 : \mathbb{Z}_m^{\mathscr{d}} \to \mathsf{PowerSet}(\mathbb{Z}_m^{\mathscr{d}})$ and $H_2 : \mathbb{Z}_m^{\mathscr{d}} \to \mathsf{PowerSet}(\mathbb{Z}_m^{\mathscr{d}})$ as:*

$$H_1(i) = \{B(j) | j \in \mathbb{Z}_m^{\mathscr{d}}, d(i,j) \leq \delta\}$$
$$H_2(i) = \{B(i)\}$$

H_1 outputs all bins that contain some point within δ of the input point. H_2 only outputs the bin its input point is in.

Claim. H_1 and H_2 are a conditionally-overlapping hash pair for the similarity function defined by $f(i,j) = d(i,j) \leq \delta$.

Proof. H_2 never outputs a set with more than 1 element. Because the cubes have length 2δ, Only bins in two "rows" of each dimension can be within δ of a particular item[3]. So, H_2 can never output a set with more than $2^{\mathscr{d}}$ items. Finally, if $d(i,j) \leq \delta$ for points i and j, the bucket in $H_2(j)$ is in $H_1(i)$ by definition.

It is also possible to divide responsibility for searching the dimensions between each party, rather than having one party be responsible for searching over every dimension. This will be useful for balancing message costs.

Definition 8. *Let $\mathbb{Z}_m$ be the ring of integers mod m for some positive integer m, $\mathscr{d}, \delta, s$ be positive integers such that $s < \mathscr{d}$, and d be the Euclidean distance between two points. Define $H_3 : \mathbb{Z}_m^{\mathscr{d}} \to \mathsf{PowerSet}(\mathbb{Z}_m^{\mathscr{d}})$ and $H_4 : \mathbb{Z}_m^{\mathscr{d}} \to \mathsf{PowerSet}(\mathbb{Z}_m^{\mathscr{d}})$ as the following:*

$$H_3(i) = \{j \in H_1(i) |\ \forall k \in \{l\}_{l=1}^{s}, j[k] = B(i)[k]\}$$
$$H_4(i) = \{j \in H_1(i) |\ \forall k \in \{l\}_{l=s+1}^{\mathscr{d}}, j[k] = B(i)[k]\}$$

H_3 includes bins close to the input that are in the same "row" as the input's bucket for the first s dimensions, and H_4 includes bins close to the input that are in the same "row" as the input's bucket for the last $\mathscr{d} - s$ dimensions.

Claim. H_3 and H_4 are a conditionally-overlapping hash pair over the function defined by $f(i,j) = [d(i,j) \leq \delta]$.

Proof. Let $i, j \in \mathbb{Z}_m^{\mathscr{d}}$ such that $d(i,j) \leq \mathscr{d}$. Consider the point $k = (i[1], ..., i[s], j[s+1], ..., j[\mathscr{d}]) = (i[l]_{l=1}^{s}, j[l]_{l=s+1}^{\mathscr{d}})$. Note that $k \in \mathbb{Z}_m^{\mathscr{d}}$.

$$d(i,k) = (\sum_{l=1}^{\mathscr{d}} (i[l] - k[l])^p)^{\frac{1}{p}} = (\sum_{l=s+1}^{\mathscr{d}} (i[l] - k[l])^p)^{\frac{1}{p}}$$

[3] If an item is in the exact middle of a bucket in some dimension, items on the edge of the bucket on both sides in this dimension can be δ away from the item. In this case, however, the smaller item is in the same bucket as the initial item.

$$= (\sum_{l=s+1}^{\mathscr{d}} (i[l] - j[l])^p)^{\frac{1}{p}} \leq (\sum_{l=1}^{\mathscr{d}} (i[l] - j[l])^p)^{\frac{1}{p}} = d(i, j),$$

so k is close to i and $B(k) \in H_1(i)$. Similarly,

$$d(j, k) = (\sum_{l=1}^{\mathscr{d}} (j[l] - k[l])^p)^{\frac{1}{p}} = (\sum_{l=1}^{s} (j[l] - k[l])^p)^{\frac{1}{p}}$$

$$= (\sum_{l=1}^{s} (j[l] - i[l])^p)^{\frac{1}{p}} \leq (\sum_{l=1}^{\mathscr{d}} (j[l] - i[l])^p)^{\frac{1}{p}} = d(j, i),$$

so k is close to j and $B(k) \in H_1(j)$. $B(k)$ is the same as $B(i)$ for the first s dimensions, and $B(k)$ is the same as $B(j)$ for the last $\mathscr{d} - s$ dimensions, so $B(k) \in H_3(i)$ and $B(k) \in H_4(j)$. Therefore, $H_3(i) \cap H_4(j) \neq \emptyset$.

H_3 can include bins in two rows for each "included" dimension, but is limited to including bins in one row for each non-included dimension, so H_3 outputs a set of no more than 2^s bins. Similarly, H_4 outputs a set of no more than $2^{\mathscr{d}-s}$ bins.

4.2 Yao's protocol

Our suggested instantiation of the proximity subprotocol is based on Yao's protocol. We formalize the necessary conditions for Yao's protocol to satisfy our pseudorandomness ssecurity properties. We use the *garbling scheme* abstraction of Bellare, Hoang, and Rogaway [BHR12], with some small modifications to both the syntax and security definitions, which we describe below:

Definition 9. *A* ***garbling scheme for boolean output*** *consists of the following algorithms:*

- *Gb: on input a circuit description f (with a single output wire), produces (F, e, d), where F is a* ***garbled circuit****, e is* ***encoding information****, d is* ***decoding information****.*
- *En: on input encoding information e and circuit input x, produces* ***garbled input*** *X. When the scheme is* ***projective****, e is structured as a $2 \times n$ matrix (where n is the input length of the circuit), and En is defined as:*

$$\mathsf{En}(e, x) = \big(e[1, x[1]], \ldots, e[n, x[n]]\big).$$

- *Ev: on input a garbled circuit F and garbled input X, produces* ***garbled output*** *Y.*

Unlike the standard definitions, we do not have a separate decoding algorithm. Instead, we demand the following correctness property: For all (F, e, d) generated by $\mathsf{Gb}(f)$, and all x, we have

$$\mathsf{Ev}(F, \mathsf{En}(e, x)) = d \text{ if and only if } f(x) = 1.$$

The reader may wish to think of d as the label that encodes `true` on the output wire. The evaluator can check whether the circuit outputs `true` by comparing its garbled output to d. We require such a scheme to satisfy the following requirement:

Definition 10. *A garbling scheme for boolean output is* ***secure*** *with respect to a class $\mathcal{G}$ of circuits if the following oracles are indistinguishable:*

```
REAL-VIEW$(f, x)$:
    assert $f \in \mathcal{G}$
    $(F, e, d) \leftarrow \mathsf{Gb}(f)$
    $X \leftarrow \mathsf{En}(e, x)$
    return $(F, X, d)$
```

$\approx$

```
SIM-VIEW$(f, x)$:
    assert $f \in \mathcal{G}$
    $(F, X) \leftarrow \{0,1\}^m$
    if $f(x) = 1$: $d := \mathsf{Ev}(F, X)$
    else: $d \leftarrow \{0,1\}^n$
    return $(F, X, d)$
```

In other words, (1) the garbled circuit and garbled input are pseudorandom, and (2) if the circuit outputs 0 then the decoding information is pseudorandom too. This definition captures several standard qualitative security properties of garbling schemes: *Privacy:* (F, X, d) reveal no more than the circuit output (and nothing about the choice of f within the class $\mathcal{G}$); *(One-sided) Authenticity:* When the circuit output is `false`, it is hard for an evaluator to guess the garbled output that encodes `true` (i.e., d). This authenticity property is expressed in the real-vs-random style (adversary cannot distinguish d from a uniform value). Importantly, this security property implies that an adversary cannot distinguish between a correct garbling that happens to output `false`, and random junk.

The protocol. When combined with a 2-message oblivious transfer (OT) protocol in the natural way, we get the variant of Yao's protocol shown in Figure 3.

```
Party 1 (input $x$)                                  Party 2 (input $y$)
**for each** $i \in \{1,\ldots,|x|\}$ :
    $M_1[i], \sigma[i] \leftarrow \mathsf{OT}_1(x[i])$
                          $\xrightarrow{M_1}$
                                                     $(F, e, d) \leftarrow \mathrm{Gb}(f_y)$
                                                     **for each** $i \in \{1,\ldots,|x|\}$ :
                                                         $T[i] \leftarrow \mathsf{OT}_2\big(M_1[i], (e[i,0], e[i,1])\big)$
                          $\xleftarrow{T, F, d}$
**for each** $i \in \{1,\ldots,|x|\}$ :
    $X[i] := \mathsf{OT}_3(T[i], \sigma[i])$
if $\mathsf{Ev}(F, X) == d$ :
    output $\{x\}$
else: output $\emptyset$
```

Fig. 3. Yao's protocol in our abstraction. The receiver has x and sender has y; the receiver will output $\{x\}$ if $f_y(x) = 1$ and $\emptyset$ otherwise.

Lemma 1. *Let $\mathcal{G} = \{f_y\}_y$ be a class of boolean-output circuits and define*

$$f(x, y) = \begin{cases} \{x\} & \text{if } f_y(x) = \texttt{true} \\ \emptyset & \text{if } f_y(x) = \texttt{false} \end{cases}.$$

The protocol in figure 3 is a pseudorandom and f-disjoint pseudorandom subprotocol for functionality f if:

- $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev})$ *is a secure (with respect to $\mathcal{G}$), projective garbling scheme for boolean output,*
- $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_3)$ *is a secure pseudorandom protocol for 1-out-of-2 OT with the additional property that its 2nd protocol messages are pseudorandom when the OT sender's input is random — i.e., the following oracles are indistinguishable:*

```
OT2-REAL():
    $M_1 \leftarrow \mathcal{M}_1$
    $v \leftarrow \mathcal{I}$
    // OT sender's input is a pair of strings
    $M_2 \leftarrow \mathsf{OT}_2(M_1, (v, v))$
    return $(M_1, M_2)$
```

$\approx$

```
OT2-RAND():
    $M_1 \leftarrow \mathcal{M}_1$
    $M_2 \leftarrow \mathcal{M}_2$
    return $(M_1, M_2)$
```

Proof (sketch). The first protocol message is a collection of OT messages, so it is pseudorandom if the OT protocol is. The more interesting property is whether second protocol messages are pseudorandom: (1) when the first message is uniform, and (2) when the first message is honestly generated but $f(x, y) = \emptyset$.

In the first case (first protocol message is uniform), we can use the following sequence of hybrids:

– Hybrid 0: The first protocol message is uniform, and second protocol message computed honestly in response. T are OT responses, F is a garbled circuit, and d is the decoding information.
– Hybrid 1: Same as above, but T is replaced with random messages. The change is indistinguishable by the psuedorandomness of the OT subprotocol.
– Hybrid 2: Same as above, but replace F and d with random messages. The change is indistinguishable by the security of the garbling scheme. More precisely, we consider a reduction algorithm in the garbling security game who chooses f_y and x such that $f_y(x) = 0$. Then F and d (and indeed X, which this reduction can ignore) are pseudorandom.

For the other case, we can use the following sequence of hybrids. We assume that x and y are such that $f_y(x) = 0$.

– Hybrid 0: The first protocol message is computed using x, and second protocol message computed honestly in response. T are OT responses, F is a garbled circuit, and d is the decoding information.
– Hybrid 1: Same as above, except the OT responses T are computed using $(e[i, x_i], e[i, x_i])$ rather than $(e[i, 0], e[i, 1])$ as the sender's input. The change is indistinguishable by the standard semi-honest security of the OT subprotocol, since the receiver's ideal OT output is $e[i, x_i]$ in both cases.
– Hybrid 2: Same as above, except F, d, and each of the $e[i, x_i]$ values is sampled uniformly. The change is indistinguishable by the security of the garbling scheme. The $e[i, x_i]$ values comprise the garbled input X in the garbling security game.
– Hybrid 3: Same as above, except that the OT responses are replaced with random messages. This change is indistinguishable by the special property of the OT protocol, since the OT sender's inputs are of the form (v, v) where v is uniform.

4.3 Instantiating Yao's protocol for distance comparison

We would like a suitable subprotocol for testing whether two private points are within distance δ.

The main idea is to garble a circuit that has the garbler's point y and distance threshold δ hard-coded. The circuit tests whether the input point is within distance δ of y and outputs a boolean. In the terminology of the previous section, what we want is a garbling scheme supporting the following class of circuits:

Definition 11. *Fix a dimension $\mathscr{d}$, and let d_p denote the ℓ_p distance function over $\mathbb{Z}^{\mathscr{d}}$. Define $f_{y,p,\mathscr{d},\delta} : \mathbb{Z}^{\mathscr{d}} \to \{0, 1\}$ to be the distance threshold function $f_{y,p,\mathscr{d},\delta}(x) = [d_p(x, y) \leq \delta]$. Let $\mathcal{G}_{p,\mathscr{d},\delta}$ be the class of functions*

$$\mathcal{G}_{p,\mathscr{d},\delta} = \{f_{y,p,\mathscr{d},\delta} \mid y \in \mathbb{Z}^{\mathscr{d}}\}$$

We require a garbling scheme that can support this class of circuits. Importantly, the garbled circuit must hide the choice of circuit from the class $\mathcal{G}_{p,\mathscr{d},\delta}$ — i.e., it must hide the choice of y. We consider different choices of distance metrics below:

Lemma 2. *The garbling scheme of Ball, Malkin, Rosulek [BMR16] satisfies our desired security definition with respect to $\mathcal{G}_{2,\mathscr{d},\delta}$ above — i.e, ℓ_2 distance. If the length (in bits) of each coordinate of the points is u bits, then the size of the garbled circuit and garbled input is $O((u\mathscr{d} + u^3)\lambda)$.*

Proof. The BHR garbling scheme expresses bounded integers in terms of their residues mod 2, mod 3, mod 5, and so on, using the Chinese remainder theorem. The scheme supports free addition mod p and arbitrary unary operations on mod-p values at a cost of $O(p\lambda)$. Importantly, the garbled circuit hides the truth table of the unary function. We can compute the necessary distance threshold using these fundamental operations in the following way:

Given input $x = (x_1, \dots, x_{\mathscr{d}})$, with each x_i represented in the residue system, we first compute $(x_i - y_i)^2$ for each i. Since this operation is algebraic, we can compute this expression in each prime modulus independently. The cost is a single unary gate (with y obliviously hard-coded) in each modulus. After this, computing $\sum_i (x_i - y_i)^2$ is free. What remains is to compute the comparison between this sum and δ^2; this is supported in the BHR scheme at a cost of $O(u^3\lambda)$.

The fact that the BHR scheme satisfies our slightly modified security definition is implicit in their proof. First, the garbled circuit and garbled inputs are explicitly sampled uniformly in their simulator. Second, their proof of authenticity involves a hybrid in which output labels other than the ones expected to be seen by the evaluator are sampled uniformly. This implies the property that is needed in our setting; namely, the `true` output label is indistinguishable from a random value, given a garbled input that evaluates to `false`.

For ℓ_1 and ℓ_∞ distance, traditional boolean garbling is a better choice:

Lemma 3. *The half-gates garbling scheme of Zahur, Rosulek, Evans [ZRE15] satisfies our security definition with respect to $\mathcal{G}_{1,\mathscr{d},\delta}$ and $\mathcal{G}_{\infty,\mathscr{d},\delta}$, and has a garbled circuit and input size of $O(u\mathscr{d}\lambda)$ for both circuit classes, where u is the length of each coordinate of each point.*

Proof. The security justification for half-gates garbling is exactly the same as the justification for the arithmetic garbling construction in the previous proof. Namely, their security proof explicitly generates the garbled circuit and encoding information uniformly, and their proof of authenticity implicitly establishes that the inactive (true) output label is pseudorandom when the evaluator is entitled to learn the false output label.

The ℓ_1 distance comparison circuit checks

$$\sum_{i=1}^{\mathscr{d}} |x[i] - y[i]| \leq \delta.$$

This circuit requires an addition (subtraction) and absolute value for each dimension, and one comparison. An addition circuit costs 1 AND gate per bit of integer. An absolute value circuit costs 2 AND gates per bit, while a comparison costs the same as an addition circuit.

The ℓ_∞ distance circuit computes:

$$\bigwedge_{i=1}^{\mathscr{d}} \Big[|x[i] - y[i]| \leq \delta \Big]$$

The cost is 4 AND gates per bit (1 for the subtraction, 2 for the absolute value, 1 for the comparison), plus 1 AND gate per dimension (for the disjunction).

A garbled AND gate costs $O(\lambda)$ bits, giving the stated communication costs.

4.4 Garbling Cost

δ Maximum Value. The arithmetic garbling we use for ℓ_2 distance supports integer arithmetic with a fixed upper bound. Thus, it is necessary to bound the maximum value of the expression $\sum_{i=1}^{\mathscr{d}} (A[i] - B[i])^2$. For any A, B in the same bin, we have $|A[i] - B[i]| \leq 4\delta$. Thus, the sum is at most $\mathscr{d}(4\delta)^2$. The garbling of scheme of [BMR16] supports integers bounded between $\pm\lfloor (Z-1)/2 \rfloor$. For a given choice of Z, we can therefore support δ such that $\mathscr{d}(4\delta)^2 \leq \lfloor (Z-1)/2 \rfloor$. Solving for δ gives an approximate maximum of $\delta \leq \sqrt{2^{u-5}/\mathscr{d}}$.

Table 2. Approximate maximum of δ for ℓ_2

$\mathscr{d}$	integer bitlength (u)		
	16	32	64
2	32	8192	536870912
5	20	5181	339546978
10	14	3663	240095970

Because ℓ_1 and ℓ_∞ do not require squaring, the maximum size of intermediate values is roughly as large as the input values. As a result, the subprotocols for ℓ_1 and ℓ_∞ support much higher values of δ than the subprotocol for ℓ_2 using the same number of bits. The threshold for ℓ_∞, specifically, also does not depend on $\mathscr{d}$, because the results of each per-dimension subtraction are not summed together. For a garbled circuit using u bits to express integers in two's complement form, input integers can only be $u-1$ bits long [4]. Input integers are no greater than 4δ, so the maximum value of δ for ℓ_∞ is simply $\delta \leq 2^{u-3}$. The ℓ_1 circuit does sum each per-dimension result together. Otherwise, the analysis is the same as for ℓ_1, so the maximum value of δ is $\delta \leq \frac{2^{u-3}}{\mathscr{d}}$.

[4] Input integers are exclusively positive, but a negative bit must be used to hold possible negative intermediate integers before the absolute value calculation

Table 3. Approximate maximum of δ for ℓ_1

d	integer bitlength (u)		
	16	32	64
2	4096	268435456	$\approx 10^{18}$
5	1638	107374182	$\approx 10^{17}$
10	819	53687091	$\approx 10^{17}$

Communication costs. The cost for a single execution for the subprotocol can be calculated based on figure 3 in [BMR16] as the sum of ciphertexts needed for OTs, squaring gates, and the comparison gate. Party 1's message cost is only the OT costs, which are the same for all distance metrics.

Table 4. Cost per subprotocol execution (in # of ciphertexts)

	Integer bitlength (u)		
	16	32	64
Party 1	$22d$	$37d$	$72d$
Party 2, ℓ_1	$96d$	$192d$	$384d$
Party 2, ℓ_2	$73d + 804$	$156d + 2541$	$437d + 11979$
Party 2, ℓ_∞	$97d - 1$	$193d - 1$	$385d - 1$

Assuming the parties use a polynomial OKVS, the total size of a party's message in the main protocol is simply the size of that party's subprotocol message[5], multiplied by the number of terms in the polynomial, the later being $h_1 \cdot |A| + 1$ for party 1 and $h_2 \cdot |B| + 1$ for party 2. The following tables assume the ciphertexts are 128 bits long.

Table 5. FPSI Message Size, ℓ_1

			Integer bitlength(u)					
d	Input set size		16		32		64	
	Party 1	Party 2	Party 1	Party 2	Party 1	Party 2	Party 1	Party 2
2	2^{20}	2^{11}	772 MB	34 MB	1.24 GB	67 MB	2.4 GB	134 MB
2	2^{16}	2^{16}	185 MB	268 MB	310 MB	537 MB	604 MB	1.07 GB
10	2^{11}	2^{11}	461 MB	671 MB	778 MB	1.3 GB	1.5 GB	2.7 GB

[5] The polynomial has to be large enough to fit encodings of protocol messages and bin descriptions. Unless the protocol is run with extraordinarily large integers, the protocol messages will be larger.

Table 6. FPSI Message Size, ℓ_2

			Integer bitlength(u)					
d	Input set size		16		32		64	
	Party 1	Party 2	Party 1	Party 2	Party 1	Party 2	Party 1	Party 2
2	2^{20}	2^{11}	738 MB	125 MB	1.2 GB	374 MB	2.4 GB	1.7 GB
2	2^{16}	2^{16}	185 MB	996 MB	310 MB	3 GB	604 MB	13.5 GB
10	2^{11}	2^{11}	923 MB	402 MB	1.6 GB	1.1 GB	3 GB	4.3 GB

The costs for ℓ_∞ are very close to the costs for ℓ_1, so the table for ℓ_∞ is omitted. In these tables, the costs are roughly balanced by changing how many dimensions each party searches over. For example, in the last row for ℓ_2, party 1 searches over 6 dimensions and party 2 searches over 4, so $h_1 = 2^6$ and $h_2 = 2^4$.

These concrete values show that our protocol can support very high values of δ in low dimensions efficiently. For example, for ℓ_1, we get a total concrete communication cost of 386 MB with $|A| = |B| = 2^{16}$, $d = 2$, and $\delta = 4096$, and a total concrete communication cost of 847 MB with $|A| = |B| = 2^{16}$, $d = 2$, and $\delta = 268435456$.

Comparison. Finally, we provide a comparison of our concrete communication costs with those of other FPSI schemes. Costs for Gao, Qi, Liu, et al. [GQL$^+$24] and Baarsen, Pu [vP24] are taken from table 4 of [GQL$^+$24]. We show costs for the low-dimensional protocol of [vP24].

Table 7. Concrete Communication Cost Comparison (in MB)

Norm	Set size	Protocol	$(\mathscr{d}, \delta)$					
			$(2, 10)$	$(6, 10)$	$(10, 10)$	$(2, 30)$	$(6, 30)$	$(10, 30)$
	2^8	BP24 [vP24]	2.85	132	3520	8.51	396	$> 10^4$
		GQL$^+$24 [GQL$^+$24]	7.5	21.8	**36.4**	21.3	63.2	**105**
		ours	**1.78**	**21.3**	142	**1.78**	**21.3**	142
ℓ_1	2^{12}	BP24	45.6	2113	$> 10^4$	136	> 6000	$> 10^5$
		GQL$^+$24	120	351	**589**	340	1024	**1703**
		ours	**28.4**	**341**	2274	**28.4**	**341**	2274
	2^{16}	BP24	730	$> 10^4$	$> 10^5$	2179	$> 10^4$	$> 10^6$
		GQL$^+$24	1919	5685	**9427**	5513	16382	**27253**
		ours	**455**	**5457**	36390	**455**	**5457**	36390
	2^8	BP24	**3.55**	132	3521	15.3	403	$> 10^4$
		GQL$^+$24	7.59	**22**	**36.9**	21.4	63.3	**107**
		ours	4.63	27.5	158	**4.63**	**27.5**	158
ℓ_2	2^{12}	BP24	**56.9**	2124	$> 10^4$	245	> 6000	$> 10^5$
		GQL$^+$24	122	**357**	**591**	347	1026	**1706**
		ours	73.8	440	2531	**73.8**	**440**	2531
	2^{16}	BP24	**911**	$> 10^4$	$> 10^5$	3919	$> 10^4$	$> 10^6$
		GQL$^+$24	1964	**5707**	**9449**	5549	16419	**27289**
		ours	1181	7034	40500	**1181**	**7034**	40500
	2^8	BP24	2.77	132	3520	8.27	396	$> 10^4$
		GQL$^+$24	7.52	22.1	**36.8**	21.4	63.9	**106**
		ours	**1.77**	**21.2**	142	**1.77**	**21.2**	142
ℓ_∞	2^{12}	BP24	44.3	2112	$> 10^4$	132	> 6000	$> 10^5$
		GQL$^+$24	120	354	**588**	343	1022	**1702**
		ours	**28.3**	**340**	2265	**28.3**	**340**	2265
	2^{16}	BP24	708	$> 10^4$	$> 10^5$	2116	$> 10^4$	$> 10^6$
		GQL$^+$24	1924	5665	**9408**	5488	16358	**27228**
		ours	**453**	**5436**	36239	**453**	**5436**	36239

Asymptotic Complexity. The communication complexity of each party is the complexity of its subprotocol multiplied by the maximum active bins of that party. The complexity of the arithmetic and garbled circuit subprotocols is discussed in section 4.3, and includes an input integer bitlength parameter u. Here, u is described based on how it scales with $\mathscr{d}$ and δ, as discussed earlier in this section. Specifically, $u = O(\log(\mathscr{d}\delta))$ for ℓ_1 and ℓ_2, and $u = O(\log(\delta))$ for ℓ_∞.

Table 8. Asymptotic Complexity Comparison. s is an integer of the parties' choice between 0 and $\mathscr{d}$.

Norm	Assumption	Protocol	Communication
ℓ_1	$\min > 2\delta(\mathscr{d}+1)$	BP24 [vP24]	$O(\delta 2^{\mathscr{d}}\mathscr{d}\lvert A\rvert + \delta\lvert B\rvert)$
	LSH	BP24	$O(\delta\mathscr{d}\lvert A\rvert^2 + \delta\lvert A\rvert\lvert B\rvert log(\lvert A\rvert))$
	disj. proj	GQL$^+$24 [GQL$^+$24]	$O((\delta\mathscr{d} + log(\delta))\lvert B\rvert + \delta\mathscr{d}\lvert A\rvert)$
	disj. hash	ours	$O(\mathscr{d}\log(\mathscr{d}\delta)(\lvert A\rvert 2^s + \lvert B\rvert 2^{\mathscr{d}-s}))$
ℓ_2	$\min > 2\delta(\sqrt{\mathscr{d}}+1)$	BP24	$O(\delta 2^{\mathscr{d}}\mathscr{d}\lvert A\rvert + \delta^2\lvert B\rvert)$
	LSH	BP24	$O(\delta\mathscr{d}\lvert A\rvert^3 + \delta^2\lvert A\rvert\lvert B\rvert^2 log(\lvert A\rvert))$
	disj. proj	GQL$^+$24	$O((\delta\mathscr{d} + log(\delta))\lvert B\rvert + \delta\mathscr{d}\lvert A\rvert)$
	disj. hash	ours	$O(\lvert A\rvert\mathscr{d}2^s\log(\mathscr{d}\delta) + \lvert B\rvert 2^{\mathscr{d}-s}(\log(\mathscr{d}\delta)\mathscr{d} + \log(\mathscr{d}\delta)^3))$
ℓ_∞		GGM24 [GGM24]	$\mathrm{O}(\mathscr{d}\lvert A\rvert log(\delta) + \lvert B\rvert log(\delta)^{\mathscr{d}})$
	$\min > 2\delta$	BP24	$O(\delta\mathscr{d}\lvert A\rvert + 2^{\mathscr{d}}\lvert B\rvert)$
	disj. proj	BP24	$O((\delta\mathscr{d})^2\lvert A\rvert + \lvert B\rvert)$
	disj. proj	GQL$^+$24	$O(\delta\mathscr{d}(\lvert A\rvert + \lvert B\rvert))$
	disj. hash	ours	$O(\mathscr{d}\log(\delta)(\lvert A\rvert 2^s + \lvert B\rvert 2^{\mathscr{d}-s}))$

5 Optimizations

5.1 Stateful Hashing

A stateful hashing scheme can be used to allow party 2 to have multiple items which hash to the same bin, without incurring extra costs.

Definition 12. *Let $\mathbb{Z}_m$ be the ring of integers mod m for some positive integer m, $\mathscr{d}$ and l_2 be positive integers, and H_3, H_4 be the hash pair defined in section 4.1 (with $\mathscr{d}$ as the dimension and aribtrary other parameters). Define $H_5 : \mathbb{Z}_m^{\mathscr{d}} \to \mathsf{PowerSet}(\mathbb{Z}_m^{\mathscr{d}}) \times \mathbb{Z}_{l_2}$ as the following:*

$$H_5(i) = H_3(i) \times \mathbb{Z}_{l_2}$$

H_6 is defined as a stateful function from $\mathbb{Z}_m^{\mathscr{d}}$ to $\mathsf{PowerSet}(\mathbb{Z}_m^{\mathscr{d}}) \times \mathbb{Z}$ that does the following:

```
H_6(i):
  S := ∅
  for each b ∈ H_4(i)
    Add (b, n[b] + 1) to S
  Output S
```

Where $n[b]$ is the number of times H_6 has previously hashed an item to a sub-bin[6] of b.

[6] A sub-bin of b is a bin b' such that $b' = (b, z)$ for some $z \in \mathbb{Z}$.

Claim. if H_4 never hashes more than l_2 items in a set B to the same bin, and H_6 only hashes the items in B, H_5 and H_6 are a conditionally-overlapping hash pair over the similarity function of (H_3, H_4)

Proof. Because H_4 only hashes up to l_2 items to a single bin, H_6 will never hash an item to a sub-bin with an index larger than l_2. Because H_5 hashes each item to the sub-bin at each index, H_5 and H_6 will hash an item to a common bin if H_3 and H_4 do.

When using the (H_5, H_6) hash pair, party 1 can reuse the same OT message for each sub-bin of an item and bin. This makes party 1's communication cost the same as when using (H_3, H_4), and party 2's cost is the same regardless. This idea of "sub-bins" can also be extended to allow party 1 to have multiple items in the same bin, although party 2 will have increased communication costs. When used for FPSI, this hash pair leaks more information than (H_3, H_4). Party 2 can hide some information about its items by randomly permuting which item it puts in each sub-bin.

5.2 Subprotocols with shared state (OT extension)

Our main protocol invokes many *completely indepedent* instances of its subprotocols, one for each bin. Our suggested subprotocol is the variant of Yao's protocol described in Section 4.2. Each instance of Yao's protocol requires many instances of oblivious transfer (OT).

Given that many OTs are needed, suppose we would like to use the technique of **OT extension** [Bea96,IKNP03]. In OT extension, the parties perform λ "expensive" *base* instances of OT; after this initial step, they can then obtain any number of effective OT instances which require only "cheap" symmetric-key operations. Within our current framework, each instance of Yao's protocol is independent, and must perform its own "expensive" base OTs.

A preferable solution would involve one *global* set of base OTs, which all of the Yao subprotocol instances could leverage. In the remainder of this section, we describe how to incorporate this optimization into our fuzzy PSI framework.

Shared state between subprotocols. The base OTs represent global shared state between subprotocol instances. Thus, we require subprotocols which are safe to use with such state. We abstract the shared state via a randomized function Setup which deals correlated randomness to the parties. Concretely, Setup represents an ideal functionality that performs λ base OTs. In the final protocol, Setup would be replaced by a (plain) secure protocol; the composition needed here is standard and trivial.

We next modify the syntax and security of the subprotocols. Suprotocols must take as input the party's shared state, along with a nonce/tag (which will be the identity of the bin in our construction). The security properties that we introduced in section 2.3 must now hold for many instances of the subprotocol, with globally shared state, provided that each nonce/tag is used only once. We obtain modified security definitions defined by the following pairs of (stateful) oracles:

```
$(A, B) \leftarrow \mathsf{Setup}()$

$\underline{\text{PROT}_1\text{-REAL}(a, \beta)}$:
    assert $\beta$ never used before
    $(M_1, \sigma) \leftarrow \mathsf{PROT}_1(\underline{A, \beta}; a)$
    Output $(B, M_1)$
```

$\approx$

```
$(A, B) \leftarrow \mathsf{Setup}()$

$\underline{\text{PROT}_1\text{-RANDOM}(a, \beta)}$:
    assert $\beta$ never used before
    $M_1 \leftarrow \mathcal{M}_1$
    Output $(B, M_1)$
```

```
$(A, B) \leftarrow \mathsf{Setup}()$

$\underline{\text{PROT}_2\text{-REAL}(b, \beta)}$:
    assert $\beta$ never used before
    $M_1 \leftarrow \mathcal{M}_1$
    $M_2 \leftarrow \mathsf{PROT}_2(\underline{B, \beta}; M_1, b)$
    Output $(A, M_1, M_2)$
```

$\approx$

```
$(A, B) \leftarrow \mathsf{Setup}()$

$\underline{\text{PROT}_2\text{-RANDOM}(b, \beta)}$:
    assert $\beta$ never used before
    $M_1 \leftarrow \mathcal{M}_1$
    $M_2 \leftarrow \mathcal{M}_2$
    Output $(A, M_1, M_2)$
```

```
$(A, B) \leftarrow \mathsf{Setup}()$

$\underline{\text{PROT}_2\text{-REAL}(a, b, \beta)}$:
    assert $\beta$ never used before
    assert $f(a, b) = \emptyset$
    $(M_1, \sigma) \leftarrow \mathsf{PROT}_1(\underline{A, \beta}; a)$
    $M_2 \leftarrow \mathsf{PROT}_2(\underline{B, \beta}; M_1, b)$
    Output $(A, M_1, \sigma, M_2)$
```

$\approx$

```
$(A, B) \leftarrow \mathsf{Setup}()$

$\underline{\text{PROT}_2\text{-RANDOM}(a, b, \beta)}$:
    assert $\beta$ never used before
    assert $f(a, b) = \emptyset$
    $(M_1, \sigma) \leftarrow \mathsf{PROT}_1(\underline{A, \beta}; a)$
    $M_2 \leftarrow \mathcal{M}_2$
    Output $(A, M_1, \sigma, M_2)$
```

In each security game, the adversary also is allowed to see its own shared state (either A or B depending on who the security property applies to).

Incorporating OT extension into Yao's protocol. In our setting, parties require *potential* OT instances designated for each bin, of which there may be exponentially many. Thus, we need an OT extension method with *random access* to specific extended OTs. This can be achieved by modifying the standard IKNP [IKNP03] protocol in a natural way.

In IKNP, parties exchange short seed values using base OTs. They then use a PRG to expand these seed values into a tall matrix, where each row of the matrix gives rise to a single instance of OT. All of their communication involves matrices of the same dimension, and the information needed for the ith extended OT is always contained in the ith row of these matrices.

So instead of expanding seed values using a PRG, we can expand using a PRF. The information corresponding to the ith OT instance is derived from $\mathsf{PRF}(\mathit{seed}, i)$. Suppose that for a certain bin β, we require OT instances that have been labeled $\beta \| 1, \ldots, \beta \| n$. Then given the base OT seeds (these are the output of Setup), we expand the n rows of the IKNP matrix corresponding to rows $\beta \| 1, \ldots, \beta \| n$, and run the usual IKNP protocol, exchanging only these rows of any matrices. It is not hard to see that the IKNP protocol messages are pseudorandom, as is needed in our setting. We defer the formal analysis of this optimization to the full version.

Henecka and Schneider previously proposed using a PRF to expand the IKNP matrix, for performance reasons [HS13]. Our overall method is similar to the *sparse OT* method of Pinkas et al [PRTY19] — like us, they encode an OKVS (interpolate a polynomial) over certain rows of the IKNP matrix.

References

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

[BMR16] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for Boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 565–577. ACM Press, October 2016.

[CDJ16] Chongwon Cho, Dana Dachman-Soled, and Stanislaw Jarecki. Efficient concurrent covert computation of string equality and set intersection. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 164–179. Springer, Cham, February / March 2016.

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

[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, May 2004.

[GGM24] Gayathri Garimella, Benjamin Goff, and Peihan Miao. Computation efficient structure-aware PSI from incremental function secret sharing. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VIII*, volume 14927 of *Lecture Notes in Computer Science*, pages 309–345. Springer, 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.

[GQL$^+$24] Ying Gao, Lin Qi, Xiang Liu, Yuanchao Luo, and Longxin Wang. Efficient fuzzy private set intersection from fuzzy mapping. Cryptology ePrint Archive, Paper 2024/1462, 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, August 2022.

[GRS23] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. Malicious secure, structure-aware private set intersection. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 577–610. Springer, Cham, August 2023.

[HS13] Wilko Henecka and Thomas Schneider. Faster secure two-party computation with less memory. In Kefei Chen, Qi Xie, Weidong Qiu, Ninghui Li, and Wen-Guey Tzeng, editors, *ASIACCS 13*, pages 437–446. ACM Press, May 2013.

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

[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.

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

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

[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.

[vP24] Aron van Baarsen and Sihang Pu. Fuzzy private set intersection with large hyperballs. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 340–369. Springer, Cham, May 2024.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Berlin, Heidelberg, April 2015.

A Appendix

A.1 Polynomial OKVS partial hiding

Section 2.1 states that our protocol requires an OKVS that hides random points, even if encoded over non-random points. We provide a proof that the polynomial OKVS has this property.

Proof. Let $n = |K_2| = |K_3|$. For any $V_2 \in \mathcal{V}^n$, define P as the polynomial interpolated over the union of the key-value pairs of K_1 and V_1 with the key-value pairs of K_2 and V_2. OKVS-PARTIAL-LEFT outputs P if[7] and only if[8] V_2 is sampled, which occurs with probability $\frac{1}{|\mathcal{V}|^n}$. Define $V_3 = \{P(K_3[i])\}_{i=1}^n$. OKVS-PARTIAL-RIGHT outputs P if and only if V_3 is sampled, which occurs with probability $\frac{1}{|\mathcal{V}|^n}$. So, the distributions are identical.

A.2 Full Proof of Theorem 1

Correctness. Party 1's output is a set O consisting of the subprotocol outputs for inputs $\mathcal{H}_1^{-1}(\beta, A)$ over all $\beta \in \mathcal{H}_1(A)$. First, suppose $\beta \in \mathcal{H}_1(A) \cap \mathcal{H}_2(B)$, i.e., β is an active bin for both parties. Then P is encoded over $(\beta, M[\beta])$ where $M[\beta]$ is the real subprotocol message generated by PROT_1, and Q is encoded over $(\beta, M'[\beta])$ where $M'[\beta]$ is the real subprotocol message generated by PROT_2; furthermore, due to correctness of the OKVS, the subprotocol message PROT_2 is run on, M_1, is the same as what it would receive in a real execution of the subprotocol with inputs $\mathcal{H}_1^{-1}(\beta, A)$ and $\mathcal{H}_2^{-1}(\beta, B)$. Then by correctness of the subprotocol, the output for β in PROT_3 must be $(\mathsf{PROT}_3(\sigma[\beta], M_2), \beta)$ with overwhelming probability.

Next, suppose $\beta \in \mathcal{H}_1(A) \backslash \mathcal{H}_2(B)$, i.e., β is an active bin only for party 1. Note that because $\mathcal{H}_2^{-1}(\beta, B) = \emptyset$, $\mathcal{H}_2^{-1}(\beta, A)$ and $\mathcal{H}_2^{-1}(\beta, B)$ have no similar elements. In this case, Q is encoded over some dummy elements that are pseudorandom, so $M_2 = \mathsf{Decode}(Q, \beta)$ is also pseudorandom. Then f-disjoint pseudorandomness implies that with overwhelming probability, PROT_3 on M_2 will output whatever a real execution of the subprotocol with non-similar input sets would output, so the output matches a real execution of the subprotocol with $\mathcal{H}_2^{-1}(\beta, A)$ and $\mathcal{H}_2^{-1}(\beta, B)$ as inputs, that is, $(\mathsf{PROT}_3(\sigma[\beta], M_2), \beta)$.

[7] There is only one minimal-degree polynomial that fits over the points, so interpolation always produces the same polynomial.

[8] If OKVS-PARTIAL-LEFT samples some vector with an item $V_2'[i] \neq V_2[i]$, then its output P' satisfies $P'(K_2[i]) = V_2'[i] \neq V_2[i]$ whereas $P(K_2[i]) = V_2[i]$, so P' cannot be P.

Corrupt party 1. We begin with describing party 1's view in the real execution, which we denote as $\textsc{real}(A, B)$.

```
$\underline{\textsc{real}(A, B)}$:
    for each $\beta_i \in \mathcal{H}_1(A)$:
        $(M[\beta_i], \sigma[\beta_i]) \leftarrow \mathsf{PROT}_1(\mathcal{H}_1^{-1}(\beta_i, A))$
        add $(\beta_i, M[\beta_i])$ to $S$

    $P := \mathsf{Encode}(S, |A| \cdot h_1 + 1)$

    for each $\beta_i \in \mathcal{H}_2(B)$:
        $M'[\beta_i] \leftarrow \mathsf{PROT}_2(\mathsf{Decode}(P, \beta_i), \mathcal{H}_2^{-1}(\beta_i, B))$
        add $(\beta_i, M'[\beta_i])$ to $S'$

    $Q := \mathsf{Encode}(S', |B| \cdot h_2 + 1)$

    output $(Q, \sigma)$
```

We also recall the simulator below.

```
$\underline{\textsc{sim}(A, \mathcal{F}_1(A, B), |B|)}$:

    for each $\beta_I \in \mathcal{H}_1(A)$:
        if $\mathcal{F}_1^{\beta_I}(A, B) \neq \emptyset$:
            $(M[\beta_I], M'[\beta_I], \sigma[\beta_I]) \leftarrow \mathsf{SIM}_1(\mathcal{H}_1^{-1}(\beta_I, A), \mathcal{F}_1^{\beta_I}(A, B))$
            add $(\beta_I, M'[\beta_I])$ to $S'$
        else:
            $(M[\beta_I], \sigma[\beta_I]) \leftarrow \mathsf{PROT}_1(\mathcal{H}_1^{-1}(\beta_I, A))$

    $Q := \mathsf{Encode}(S', |B| \cdot h_2 + 1)$

    output $(Q, \sigma)$
```

We now introduce a sequence of hybrids, indexed by $t = 0, \ldots, M$, that starts from the real view and ends at the simulated view. Below we use SIM_1 to denote the simulator for the subprotocol when party 1 is corrupt; its output includes a simulated first message, a simulated second message, and a simulated internal state of party 1.

```
$\underline{\text{HYBRID}_t(A, B)}$:

  for each $\beta_i \in \mathcal{H}_1(A)$:
    if $i \leq t$ and $\mathcal{F}_1^{\beta_i}(A, B) \neq 0$:
      $(M[\beta_i], M'[\beta_i], \sigma[\beta_i]) \leftarrow \mathsf{SIM}_1(\mathcal{H}_1^{-1}(\beta_i, A), \mathcal{F}_1^{\beta_i}(A, B))$
      add $(\beta_i, M'[\beta_i])$ to $S'$
    else:
      $(M[\beta_i], \sigma[\beta_i]) \leftarrow \mathsf{PROT}_1(\mathcal{H}_1^{-1}(\beta_i, A))$
    add $(\beta_i, M[\beta_i])$ to $S$

  $P := \mathsf{Encode}(S, |A| \cdot h_1 + 1)$

  for each $\beta_i \in \mathcal{H}_2(B)$:
    if $i > t$:
      $M'[\beta_i] \leftarrow \mathsf{PROT}_2(\mathsf{Decode}(P, \beta_i), \mathcal{H}_2^{-1}(\beta_i, B))$
      add $(\beta_i, M'[\beta_i])$ to $S'$

  $Q := \mathsf{Encode}(S', |B| \cdot h_2 + 1)$

  output $(Q, \sigma)$
```

If $t = 0$ then we always enter the else-clause of the first if-statement, and always enter the second if-statement, so $\text{HYBRID}_0(A, B) \equiv \text{REAL}(A, B)$.

Claim.

$$\text{HYBRID}_t(A, B) \approx \text{HYBRID}_{t+1}(A, B).$$

We prove the lemma via a sequence of intermediate sub-hybrids.

$\text{HYBRID}_t^1(A, B)$: If $\beta_{t+1} \notin \mathcal{H}_1(A)$, i.e., β_{t+1} is not an active bin for party 1, add a dummy point (β_{t+1}, M_1^*) (where $M_1^* \leftarrow \mathcal{M}_1$) to S. By the definition of expand-then-encode in OKVS, the adversary's view remains identical. Note that there are at most $|A| \cdot h_1$ active bins, so $|S| \leq |A| \cdot h_1 + 1$ even after adding the point.

$\text{HYBRID}_t^2(A, B)$: If $\beta_{t+1} \in \mathcal{H}_2(B) \setminus \mathcal{H}_1(A)$ (case 1 in the proof sketch),replace the real second subprotocol message $M'[\beta_{t+1}]$ with a random $M'[\beta_{t+1}] \leftarrow \mathcal{M}_2$. This is indistinguishable from the previous hybrid due to the second pseudorandomness of the subprotocol.

$\text{HYBRID}_t^3(A, B)$: Again if $\beta_{t+1} \in \mathcal{H}_2(B) \setminus \mathcal{H}_1(A)$, remove (β_{t+1}, M_1^*) from S and remove $(\beta_{t+1}, M'[\beta_{t+1}])$ from S'. Since M_1^* and $M'[\beta_{t+1}]$ are both random, it does not make a difference to remove these two dummy points.

Note that $\text{HYBRID}_t^3(A, B)$ is identical to $\text{HYBRID}_t(A, B)$, except that $(\beta_{t+1}, M'[\beta_{t+1}])$ is not added to S' if $\beta_{t+1} \in \mathcal{H}_2(B) \setminus \mathcal{H}_1(A)$.

HYBRID$_t^4(A,B)$: In this hybrid we deal with the case that $\beta_{t+1} \in \mathcal{H}_1(A) \cap \mathcal{H}_2(B)$, i.e., β_{t+1} is an active bin for both parties. In this case, move the code for $i = t+1$ in the second for-loop to the first for-loop. The resulting code is shown below.

```
HYBRID$_t^4(A,B)$:

  for each $\beta_i \in \mathcal{H}_1(A)$:
    if $i \le t$ and $\mathcal{F}_1^{\beta_i}(A,B) \neq \emptyset$:
      $(M[\beta_i], M'[\beta_i], \sigma[\beta_i]) \leftarrow \mathsf{SIM}_1((\mathcal{H}_1^{-1}(\beta_i, A), \beta_i), \mathcal{F}_1^{\beta_i}(A,B))$
      add $(\beta_i, M'[\beta_i])$ to $S'$
    else if $i = t+1$ and $\beta_i \in \mathcal{H}_2(B)$:
      $(M[\beta_i], \sigma[\beta_i]) \leftarrow \mathsf{PROT}_1(\mathcal{H}_1^{-1}(\beta_i, A))$
      $M'[\beta_i] \leftarrow \mathsf{PROT}_2(M[\beta_i], \mathcal{H}_2^{-1}(\beta_i, B))$
      add $(\beta_i, M'[\beta_i])$ to $S'$
    else:
      $(M[\beta_i], \sigma[\beta_i]) \leftarrow \mathsf{PROT}_1(\mathcal{H}_1^{-1}(\beta_i, A))$
    add $(\beta_i, M[\beta_i])$ to $S$

  $P := \mathsf{Encode}(S, |A| \cdot h_1 + 1)$

  for each $\beta_i \in \mathcal{H}_2(B)$:
    if $i > t+1$:
      $M'[\beta_i] \leftarrow \mathsf{PROT}_2(\mathsf{Decode}(P, \beta_i), \mathcal{H}_2^{-1}(\beta_i, B))$
      add $(\beta_i, M'[\beta_i])$ to $S'$

  $Q := \mathsf{Encode}(S', |B| \cdot h_2 + 1)$

  output $(Q, \sigma)$
```

Since β_{t+1} is in $\mathcal{H}_2(B)$, in the previous hybrid the code

$$M'[\beta_{t+1}] \leftarrow \mathsf{PROT}_2(\mathsf{Decode}(P, \beta_{t+1}), \mathcal{H}_2^{-1}(\beta_{t+1}, B));\text{ add } (\beta_i, M'[\beta_i]) \text{ to } S'$$

is run in the second for-loop, whereas now it is run in the first for-loop instead. The only difference is when the code is executed, so the two hybrids are identical.

HYBRID$_t^5(A,B)$: Again if $\beta_{t+1} \in \mathcal{H}_1(A) \cap \mathcal{H}_2(B)$, replace the real $(M[\beta_{t+1}], M'[\beta_{t+1}], \sigma[\beta_{t+1}])$ generated by PROT_1 and PROT_2 with the simulated $(M[\beta_{t+1}], M'[\beta_{t+1}], \sigma[\beta_{t+1}]) \leftarrow \mathsf{SIM}_1((\mathcal{H}_1^{-1}(\beta_{t+1}, A), \beta_{t+1}), f_1^{\beta_{t+1}}(A,B))$. This is indistinguishable due to the security of the subprotocol.

HYBRID$_t^6(A,B)$: If $\beta_{t+1} \in \mathcal{H}_1(A) \cap \mathcal{H}_2(B)$ and $\mathcal{F}_1^{\beta_i}(A,B) = 0$ (case 2 in the proof sketch), replace the simulated $(M[\beta_{t+1}], M'[\beta_{t+1}], \sigma[\beta_{t+1}]) \leftarrow \mathsf{SIM}_1((\mathcal{H}_1^{-1}(\beta_{t+1}, A), \beta_{t+1}), \mathcal{F}_1^{\beta_{t+1}}(A,B))$ with a random $M'[\beta_{t+1}] \leftarrow \mathcal{M}_2$. ($M[\beta_{t+1}]$ and $\sigma[\beta_{t+1}]$ are still generated by SIM_1.) This is indistinguishable from the previous hybrid due to the f-disjoint pseudorandomness of the subprotocol.

HYBRID$_t^7(A,B)$: Again if $\beta_{t+1} \in \mathcal{H}_1(A) \cap \mathcal{H}_2(B)$ and $\mathcal{F}_1^{\beta_i}(A,B) = 0$, remove $(\beta_{t+1}, M'[\beta_{t+1}])$ from S'. Since $M'[\beta_{t+1}]$ is random, it does not make a difference to remove this dummy point. Note that $M'[\beta_{t+1}]$ is not used anywhere in the game now.

HYBRID$_t^8(A,B)$: Again if $\beta_{t+1} \in \mathcal{H}_1(A) \cap \mathcal{H}_2(B)$ and $\mathcal{F}_1^{\beta_i}(A,B) = 0$, replace the simulated $(M[\beta_{t+1}], \star, \sigma[\beta_{t+1}]) \leftarrow \mathsf{SIM}_1((\mathcal{H}_1^{-1}(\beta_{t+1}, A), \beta_{t+1}), \mathcal{F}_1^{\beta_{t+1}}(A,B))$ with the real $(M[\beta_{t+1}], \sigma[\beta_{t+1}]) \leftarrow \mathsf{PROT}_1(\mathcal{H}_1^{-1}(\beta_{t+1}, A))$. This is indistinguishable from the previous hybrid due to the first pseudorandomness of the subprotocol.

The code for HYBRID$_t^8(A,B)$ is shown below.

```
HYBRID$_t^8(A,B)$:

for each $\beta_i \in \mathcal{H}_1(A)$:
    if $i \le t$ and $\mathcal{F}_1^{\beta_i}(A,B) \neq \emptyset$:
        $(M[\beta_i], M'[\beta_i], \sigma[\beta_i]) \leftarrow \mathsf{SIM}_1((\mathcal{H}_1^{-1}(\beta_i, A), \beta_i), \mathcal{F}_1^{\beta_i}(A,B))$
        add $(\beta_i, M'[\beta_i])$ to $S'$
    else if $i = t+1$ and $\beta_{t+1} \in \mathcal{H}_2(B)$ and $\mathcal{F}_1^{\beta_i}(A,B) \neq \emptyset$:
        $(M[\beta_i], M'[\beta_i], \sigma[\beta_i]) \leftarrow \mathsf{SIM}_1((\mathcal{H}_1^{-1}(\beta_i, A), \beta_i), \mathcal{F}_1^{\beta_i}(A,B))$
        add $(\beta_i, M'[\beta_i])$ to $S'$
    else:
        $(M[\beta_i], \sigma[\beta_i]) \leftarrow \mathsf{PROT}_1(\mathcal{H}_1^{-1}(\beta_i, A))$
    add $(\beta_i, M[\beta_i])$ to $S$

$P := \mathsf{Encode}(S, |A| \cdot h_1 + 1)$

for each $\beta_i \in \mathcal{H}_2(B)$:
    if $i > t+1$:
        $M'[\beta_i] \leftarrow \mathsf{PROT}_2(\mathsf{Decode}(P, \beta_i), \mathcal{H}_2^{-1}(\beta_i, B))$
        add $(\beta_i, M'[\beta_i])$ to $S'$

$Q := \mathsf{Encode}(S', |B| \cdot h_2 + 1)$

output $(Q, \sigma)$
```

HYBRID$_{t+1}(A,B)$: We make two changes to the code of the game: first, the behaviors in the first two if-clauses of the first for-loop are identical, so these cases can be merged; second, because $\mathcal{F}_1^{\beta_i}(A,B) \neq 0$ implies $\beta_i \in \mathcal{H}_2(B)$, the $\beta_i \in \mathcal{H}_2(B)$ requirement can be removed from the second if-clause. The resulting game is exactly HYBRID$_{t+1}(A,B)$. This completes the proof of the claim.

Finally, we consider the last hybrid HYBRID$_M(A,B)$. In the first for-loop, $(M[\beta_i], \sigma[\beta_i])$ is simulated if $\mathcal{F}_1^{\beta_{t+1}}(A,B) \neq 0$ and real otherwise, and the second for-loop is useless as the condition $i > M$ is never met. This is exactly what the ideal world does; we conclude that HYBRID$_M(A,B) \equiv$ IDEAL(A,B). Since M is the number of bins that can be in $\mathcal{H}_1(A) \cup \mathcal{H}_2(B)$, which is polynomial in the security parameter, the total number of hybrids is polynomial.

In sum, we have shown that $\textsc{real}(A, B) \approx \textsc{ideal}(A, B)$, completing the proof.

Corrupt party 2. The simulation in this case is trivial; see proof sketch.