Distance-Aware OT with Application to Fuzzy PSI

Lucas Piske
Arizona State University
lpiske@asu.edu

Jaspal Singh
Georgia Institute of Technology
wsing1361@purdue.edu

Ni Trieu
Arizona State University
nitrieu@asu.edu

Vladimir Kolesnikov
Georgia Institute of Technology
kolesnikov@gatech.edu

Vassilis Zikas
Georgia Institute of Technology
vzikas@gatech.edu

ABSTRACT

A two-party fuzzy private set intersection (PSI) protocol between Alice and Bob with input sets A and B allows Alice to learn nothing more than the points of Bob that are "δ-close" to its points in some metric space $\texttt{dist}$. More formally, Alice learns only the set $\{b \mid \texttt{dist}(a, b) \leq \delta, a \in A, b \in B\}$ for a predefined threshold δ and distance metric $\texttt{dist}$, while Bob learns nothing about Alice's set. Fuzzy PSI is a valuable privacy tool in scenarios where private set intersection needs to be computed over imprecise or measurement-based data, such as GPS coordinates or healthcare data. Previous approaches to fuzzy PSI rely on asymmetric cryptographic primitives, generic two-party computation (2PC) techniques like garbled circuits, or function secret sharing methods, all of which are computationally intensive and lead to poor concrete efficiency.

This work introduces a new modular framework for fuzzy PSI, primarily built on efficient symmetric key primitives. Our framework reduces the design of efficient fuzzy PSI to a novel variant of oblivious transfer (OT), which we term distance-aware random OT (da-ROT). This variant enables the sender to obtain two random strings (r_0, r_1), while the receiver obtains one of these values r_b, depending on whether the receiver's input keyword a and the sender's input keyword b are close in some metric space i.e., $\texttt{dist}(a, b) \leq \delta$. The da-ROT can be viewed as a natural extension of traditional OT, where the condition (choice bit) is known to the receiver. We propose efficient constructions for da-ROT based on standard OT techniques tailored for small domains, supporting distance metrics such as the Chebyshev norm, the Euclidean norm, and the Manhattan norm.

By integrating these da-ROT constructions, our fuzzy PSI framework achieves up to a 14× reduction in communication cost and up to a 54× reduction in computation cost compared to previous state-of-the-art protocols, across input set sizes ranging from 2^8 to 2^{16}. Additionally, we extend our framework to compute fuzzy PSI cardinality and fuzzy join from traditional PSI-related functionalities. All proposed protocols are secure in the semi-honest model.

ACM CCS '25, October 2025, Taipei, Taiwan

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Private Set Intersection (PSI) is a cryptographic primitive that allows two parties to compute the intersection of their sets without revealing any information. Traditional PSI protocols focus on exact matching of elements, with many efficient and practical implementations [19, 22, 25, 27, 34]. However, these approaches are not well-suited for cases requiring approximate matches or similarity-based comparisons, such as biometric identification, DNA sequence alignment, or location-based services. In this work, we focus on fuzzy PSI, where the goal is to compute intersections based on proximity or similarity, rather than strict equality.

In recent years, fuzzy PSI protocols have been the focus of active research. However, they still face practical limitations. For example, Garimella et al. [14, 15] introduced structure-aware PSI, which is designed for scenarios where the receiver has geometric objects (e.g., balls or regions) and the sender possesses points, with the aim of identifying which points fall within these regions. This variant can implement fuzzy PSI; however, their protocols are based on function secret sharing (FSS), and thus, mainly focus on L_∞ and L_1 distance metrics. While they achieve relative communication efficiency, their computational efficiency remains limited, especially when applied to more general distance metrics or arbitrary shapes. Recent work by [11] improves the computational limitations of the prior protocols, but still supports only L_∞ metric due to its reliance on the same foundational building block of FSS.

In certain applications, such as location-based services for car sharing, the L_2 norm is often required. To address this, Baarsen and Pu [37] presented a novel protocol that supports arbitrary distance metrics. However, their protocol is based on public-key operations and has a computational complexity of $O(\delta 2^d dn + m)$, where the receiver holds n hyperballs of radius δ and the sender holds m points in $\mathbb{Z}_d$. Thus, their protocol is still computationally expensive due to the high number of public key computations required. Furthermore, their protocol only works under a disjoint assumption, meaning that each receiver's point has at least one dimension in which its component maintains a distance greater than 2δ from the other.

Gao et al. [9] improved upon [37] by presenting the first protocol that achieves linear complexity with respect to input sizes, dimensions, and radius δ. However, unlike [37], their protocol is based on a different and unrealistic disjoint projection assumption; each sender's or receiver's point is separated from other points in the same set by at least one dimension. Most recently, [31] generalizes the PSI approach of [6] to the fuzzy PSI setting by combining that approach with Gabled Circuits and spatial hashing techniques, leading to a general framework that supports L_p and L_∞ metrics. They propose the use of arithmetic garbling for the L_2, while other

metrics like L_1 and L_∞ are instantiated using regular boolean-based Garbled Circuits. Although it is based on symmetric primitives, the use of generic two-party computation leads to poor concrete efficiency for fuzzy PSI.

1.1 Our Contribution

Distance-aware OT. We introduce the notion of a **distance-aware random OT** (da-ROT) parameterized by a distance metric (dist) and threshold distance ($\delta > 0$). This variant enables the sender to obtain two random strings (r_0, r_1), while the receiver obtains one of these values r_c, depending on whether the receiver's input keyword a and the sender's input keyword b are δ-close, i.e., $\texttt{dist}(a, b) \leq \delta$. We then present new concretely efficient da-ROT protocols for Chebyshev norm (L_∞), Manhattan norm (L_1), and Euclidian norm (L_2) for small domains based on oblivious transfer (OT).

We also introduce a "sparse variant" of the da-ROT functionality, which informally allows the sender and receiver to execute multiple instances of da-ROT (each labeled with an index), while also hiding from each party which indexes are in the other party's input. We present a sparse compiler (sparse-comp) that converts a class of da-ROT protocols into their sparse variants using sparse OT [26], shared OT [28], and oblivious key-value store (OKVS) [13].

New Fuzzy PSI Framework. One of our key contributions is a new modular fuzzy PSI framework for any arbitrary L_p norm ($\mathsf{p} \in \mathbb{N} \cup \{\infty\}$) based on spatial hashing introduced by Garimella et al. [14] and the newly introduced sparse da-ROT primitive for the same distance metric. Our framework reduces the design of fuzzy PSI protocols to the design of efficient da-ROT protocols for small domains. A comparison with previous works is presented in Table 1. Previous works either use expensive public-key crypto or have very poor computational complexity - given their use of generic two-party computation or FSS-based methods. To the best of our knowledge, we present the first efficient fuzzy PSI protocol primarily based on symmetric key techniques [1]. The only assumption in our construction is that the sender input set has, at max, one input point per grid cell (where we assume the entire d-dimensional input space is partitioned into grid cells of side length 2δ). This is a weaker assumption than most other previous works based on FSS and asymmetric crypto techniques, which require some form of disjointness of input points.

Other Related Fuzzy Functionalities. In addition to our primary contributions, we demonstrate how our framework can be adapted to support other related fuzzy PSI functionalities. Specifically, we show that our framework reduces the problem of fuzzy PSI cardinality (which computes only the cardinality of the fuzzy intersection) to the well-established problem of traditional PSI cardinality, which has been extensively studied and optimized in the literature [8, 12, 16, 24]. Our framework also reduces the fuzzy join problem—where the join of two input databases is based on a fuzzy metric, and the result is secret-shared—into its non-fuzzy variant [5, 20]. To the best of our knowledge, we provide the first concrete protocol for privacy-preserving fuzzy database join for various norms. We believe that this modular approach offers a useful framework that can aid future research focused on designing protocols for fuzzy PSI-related functionalities.

Fuzzy PSI Implementation. We developed a prototype implementation of our fuzzy PSI framework using the newly proposed da-ROT protocols. Our evaluation shows that the framework achieves up to a 14× reduction in communication cost and up to a 54× reduction in computation cost compared to previous state-of-the-art protocols, across input set sizes ranging from 2^8 to 2^{16}.

1.2 Technical Overview

Gao et al. [9] highlight that most *fuzzy PSI* protocols [11, 14, 15, 37] rely on a batched fuzzy matching approach to determine whether a receiver's point $\boldsymbol{a}_i$ and a sender's point $\boldsymbol{b}_j$ satisfy $\texttt{dist}(\boldsymbol{a}_i, \boldsymbol{b}_j) \leq \delta$. The result is then revealed to the receiver. This process typically involves two distinct phases: coarse mapping and refined filtering. In the coarse mapping phase, identifiers are assigned to points from both the sender and receiver, establishing initial pairings between a receiver point $\boldsymbol{a}_i$ and a sender point $\boldsymbol{b}_j$ if they share the same identifier. In the refined filtering phase, fuzzy matching is applied to each pair formed during the coarse mapping phase, yielding the final results.

This work introduces a new random OT variant, termed distance-aware random OT (da-ROT), alongside a scheme for compiling da-ROT protocols into sparse versions. This innovation effectively bridges the gap between fuzzy PSI protocols' refined filtering and coarse mapping phases. Leveraging these two building blocks with existing sparse hashing techniques, we present an efficient and scalable fuzzy PSI protocol and computation on fuzzy PSI. The technical overview of our framework is presented in Figure 1, which illustrates the various components and their interdependencies.

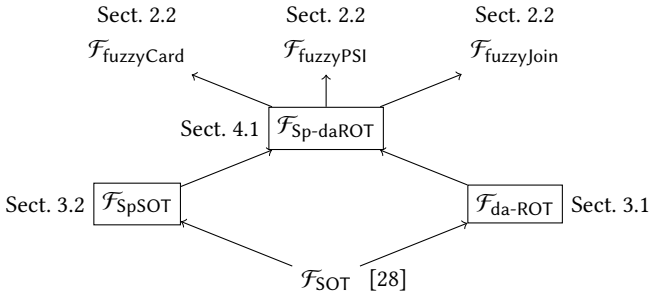

Figure 1: Technical Overview of Our *Fuzzy PSI* Framework. The new functionalities are marked with rectangles.

Our da-ROT protocol for distance metric $\mathsf{p} \in \{\infty, 1, 2\}$ starts by obliviously evaluating $z_i = f_\mathsf{p}^{(i)}(a_i)$ for every $0 \leq i \leq d$. The definition of the function f depends on b_i, δ, and the metric p. For example, for L_∞, the function $f^{(i)}$ is defined as

$$f_\infty^{(i)}(x) = \begin{cases} 1 & \text{if } x \in [b_i - \delta, b_i + \delta], \\ 0 & \text{otherwise.} \end{cases}$$

As a result of every evaluation, the two parties receive additive shares of h_i, which are then homomorphically summed to obtain

[1] Our proposed protocol relies on a small number of base OTs, which are implemented using public-key techniques. Since generating a batch of base OTs typically takes only a few hundred milliseconds per party under reasonable network conditions [23], this overhead is negligible compared to the overall protocol execution time.

Table 1: Asymptotic Complexities of Fuzzy PSI Protocols, where the receiver and sender hold n and m hyperballs of radius δ in $\mathbb{Z}^d$, respectively. All multiplicative factors of the computational security parameter κ and statistical security parameter λ are ignored.

Dist.	Prot.	Assumption	Communication	Computation		Symmetric Crypto
				Sender	Receiver	
L_∞	[11]	R. $l_{\min} > 2\delta$	$O\left(m + (4\log\delta)^d n\right)$	$O\left((2\log\delta)^d m\right)$	$O\left((2\delta)^d n\right)$	✓
	[37]	R. $l_{\min} > 2\delta$	$O\left(2^d m + \delta dn\right)$	$O\left(2^d dm\right)$	$O\left(2^d m + \delta dn\right)$	✗
		R. disj. proj.	$O\left(m + (\delta d)^2 n\right)$	$O\left(d^2 m\right)$	$O\left(m + (\delta d)^2 n\right)$	✗
	[9]	$R \wedge S$. disj. proj.	$O(\delta dm + \delta dn)$	$O(\delta dm + n)$	$O(m + \delta dn)$	✗
	[31]	disj. hash	$O(d \cdot \log(\delta)(n \cdot 2^d + m \cdot 2^{d-s}))$	$O(d\delta \cdot m2^{d-s})$	$O(d\delta \cdot n2^s)$	✓
	Ours	disj. hash, $\rho_\eta^R = 1$	$O(d \cdot (n \cdot 2^d + m \cdot \delta))$	$O(d \cdot m \cdot \delta)$	$O(d \cdot n \cdot 2^d + m)$	✓
L_1	[37]	R. $l_{\min} > 2\delta\left(d^{\frac{1}{\mathrm{p}}} + 1\right)$	$O\left(\delta^{\mathrm{p}} m + \delta 2^d dn\right)$	$O\left((d + \delta^{\mathrm{p}})\, m\right)$	$O\left(m + \delta 2^d dn\right)$	✗
		R. $l_{\min} > \frac{1}{\rho}\delta$	$O\left((\delta^{\mathrm{p}} n^\rho \log n)\, m + \delta dn^{\rho+1}\right)$	$O\left(((d + \delta^{\mathrm{p}})\, n^\rho \log n)\, m\right)$	$O\left((n^\rho \log n)\, m + \delta dn^{\rho+1}\right)$	✗
	[9]	$R \wedge S$. disj. proj.	$O((\delta d + \mathrm{p}\log\delta)m + \delta dn)$	$\lvert O((\delta d + \mathrm{p}\log\delta)m + n)\rvert$	$O(\mathrm{p}\log\delta m + \delta dn)$	✗
	[31]	disj. hash	$O(d \cdot \log(d\delta)(n \cdot 2^d + m \cdot 2^{d-s}))$	$O(\log(d\delta)d \cdot m2^{d-s} + d \cdot m2^{d-s})$	$O(\log(d\delta)d \cdot n2^s + d \cdot n2^s)$	✓
	Ours	disj. hash, $\rho_\eta^R = 1$	$O(d \cdot (n \cdot 2^d + m \cdot \delta))$	$O(d \cdot m \cdot \delta)$	$O(d \cdot n \cdot 2^d + m)$	✓
L_2	[37]	R. $l_{\min} > 2\delta\left(d^{\frac{1}{\mathrm{p}}} + 1\right)$	$O\left(\delta^{\mathrm{p}} m + \delta 2^d dn\right)$	$O\left((d + \delta^{\mathrm{p}})\, m\right)$	$O\left(m + \delta 2^d dn\right)$	✗
		R. $l_{\min} > \frac{1}{\rho}\delta$	$O\left((\delta^{\mathrm{p}} n^\rho \log n)\, m + \delta dn^{\rho+1}\right)$	$O\left(((d + \delta^{\mathrm{p}})\, n^\rho \log n)\, m\right)$	$O\left((n^\rho \log n)\, m + \delta dn^{\rho+1}\right)$	✗
	[9]	$R \wedge S$. disj. proj.	$O((\delta d + \mathrm{p}\log\delta)m + \delta dn)$	$\lvert O((\delta d + \mathrm{p}\log\delta)m + n)\rvert$	$O(\mathrm{p}\log\delta m + \delta dn)$	✗
	[31]	disj. hash	$O(nd2^s \log(d\delta) + m2^{d-s}(\log(d\delta)d + \log(d\delta)^3))$	$O((\log(d\delta^2)d + d + \log(d\delta^2))^3 \cdot m2^{d-s})$	$O((\log(d\delta^2)d + d + \log(d\delta^2))^3 \cdot n2^s)$	✓
	Ours	disj. hash, $\rho_\eta^R = 1$	$O(d \cdot (n \cdot 2^d + m \cdot \delta))$	$O(d \cdot m \cdot \delta)$	$O(d \cdot n \cdot 2^d + m)$	✓

- R. $l_{\min} > l_*$ means that the minimum distance between the receiver's points is greater than l_*.
- R. disj. proj. means that for every pair of receiver points (u, v), there must be at least one dimension i where the components u_i and v_i are further apart than 2δ.
- R $\wedge$ S. disj. proj. means that the disj. proj. assumption should hold for both the sender's input set and the receiver's input set.
- disj. hash means the spatial hashing scheme used by the construction maps at most one point to every possible target grid cell/sparse index.
- ρ_η^R represents the maximum number of receiver points in neighboring cells for any grid cell; for $\rho_\eta^R > 1$ our protocol complexity has an additional multiplicative factor ρ_η^R

additive shares of $z = h_0 + \cdots + h_{d-1}$. Assuming the modulo of the additive shares is large enough to prevent wrap-around during the computation of z, this value z has a property that enables determining whether $\mathtt{dist}(\boldsymbol{a}, \boldsymbol{b}) \le \delta$. In the case of L_∞, we have

$$z = d \iff \forall_{i\in[d]}\ a_i \in [b_i - \delta, b_i + \delta]$$

Now, to ensure the correct OT messages are sent to the corresponding parties, the parties obliviously evaluate $y = g_{\mathrm{p}}(z)$, receiving additive shares of the result. The function g depends on d, δ, and the metric p. For example, with the sender's chosen random value r, the function for the L_∞ can be defined as:

$$g_\infty(x) = \begin{cases} 0 & \text{if } x = d, \\ r & \text{otherwise.} \end{cases}$$

Clearly, $y = 0$ if $\mathtt{dist}(\boldsymbol{a}, \boldsymbol{b}) \le \delta$, and r otherwise. If we denote the sender's and receiver's additive shares of y as y^R and y^S, respectively, the receiver outputs $r_c = y^R$, while the sender defines $r_0 = -y^S$ and $r_1 = r - y^S$. Thus, we have if $y = 0$, $r_c = r_0$, otherwise, $r_c = r_1$. This ensures that the OT messages are correctly routed based on whether $\mathtt{dist}(\boldsymbol{a}, \boldsymbol{b}) \le \delta$

An important aspect that we previously overlooked is that the functions f and g must remain hidden from the receiver. Additionally, all outputs of the function evaluations are represented as additive shares, as described earlier. To achieve this, we rely on the Share OT (SOT) primitive introduced in [28]. The SOT primitive extends the functionality of 1-out-of-v OT by supporting additively secret-shared modulo-N choice indices and producing additive secret shares modulo-M of the v sender's messages as output. In our fuzzy PSI application, we work with a small value v such as $v = 2^8$, which allows us to implement this primitive efficiently. Concretely, the sender can prepare all possible values of the function f such as $m_j = f_\infty^{(i)}(j)$, for example, and act as the OT sender. Meanwhile, the receiver begins by secret-sharing the index a_i to the sender. This setup then allows the parties to use the SOT protocol to obtain the share of $f_\infty^{(i)}(a_j)$ for each j.

As a building block, we introduce a second OT variant called Sparse SOT. As input, the Sparse SOT functionality $\mathcal{F}_{\mathsf{SpSOT}}$ takes a set of tuples $(i, [\![c^{(i)}]\!]_N^R)$ from the receiver and a set of tuples $(j, [\![c^{(j)}]\!]_N^S, \boldsymbol{m}^{(j)})$ from the sender, where the first tuple components are indexes, the second are secret shares mod-N, and the third component of the sender's tuple is a vector of messages. The functionality pairs up the two parties' inputs based on their associated indexes and outputs secret shares mod-M of $m_{c^{(i)}}^{(j)}$, where $j = i$. For their respective non-paired inputs, the parties get random mod-M elements. Our protocol for $\mathcal{F}_{\mathsf{SpSOT}}$ is a natural combination of the Sparse OT Extension first introduced in [26], and the SOT protocol introduced in [28], changing the OKVS scheme used by the Sparse OT for a more efficient one.

We then use da-ROT and SpSOT to construct the sparse variant of the $\mathcal{F}_{\mathsf{da\text{-}ROT}}$ functionality, which serves as the main component of our *fuzzy PSI* protocol. The $\mathcal{F}_{\mathsf{Sp\text{-}daROT}}$ functionality takes as input a set of pairs (i, a_i) from the receiver and a set of pairs (j, b_i) from the sender, where the first component of each pair is an index and the second component is a point. The functionality matches points between the parties based on their associated indexes. For each matched pair, it performs a da-ROT operation on the points, providing the resulting outputs to the respective parties. The parties receive random outputs for points that do not have a matching counterpart (i.e., points with no index alignment). These outputs are indistinguishable from "real" da-ROT outputs, preserving the security and correctness guarantees of the da-ROT functionality.

Finally, using $\mathcal{F}_{\text{Sp-daROT}}$, spatial hashing techniques, and an OKVS, we construct protocols for *fuzzy PSI*, *fuzzy cardinality*, and computation on fuzzy matching for the metrics L_∞, L_1 and L_2.

For the *fuzzy PSI* protocol, the receiver and sender start by mapping their respective input points into indexes of an index set $\mathcal{I}$ using spatial hashing. After mapping its input point set B, the sender holds a set B' of pairs (i, b), where i is the mapped index and b is an input original point. Next, they invoke the $\mathcal{F}_{\text{Sp-daROT}}$ functionality, providing their points and associated indexes as inputs. After receiving its output message pairs $\boldsymbol{r}^{(i)} = (r_0^{(i)}, r_1^{(i)})$ from $\mathcal{F}_{\text{Sp-daROT}}$, the sender constructs an OKVS D to send to the receiver. The OKVS D is defined as:

$$D \leftarrow \text{OKVS.Encode}(Y), \text{ where } Y = \{(i, \text{Enc}(r_0^{(i)}, b)) \mid (i, b) \in B'\}.$$

Here, Enc is a symmetric encryption algorithm. After receiving D, the receiver queries it at a position j, using $r^{(j)}$ from $\mathcal{F}_{\text{Sp-daROT}}$ to decrypt, where j is an index the receiver mapped one of its points to using spatial hashing. If the receiver successfully decrypts a queried index, it implies the associated point matches one of the sender's input points.

The *fuzzy cardinality* protocols start very similarly. It maps the two parties' input points into indexes of set $\mathcal{I}$ using spatial hashing, and they invoke the $\mathcal{F}_{\text{Sp-daROT}}$ functionality, providing the mapped indexes and associated points as input. After that, however, the sender computes the set Y and the receiver computes the receiver X, where $Y = \{r_0^{(i)} \mid i \in B'\}$ and $X = \{r^{(j)} \mid j \in A'\}$. Here, B' and A' are the set of mapped indexes the sender and receiver got from spatial hashing their input points, respectively. At the same time, $r_0^{(i)}$ and $r^{(j)}$ are the outputs the sender and receiver got from the $\mathcal{F}_{\text{Sp-daROT}}$ for every respective index, respectively. After assembling these sets, the two parties invoke a regular *PSI cardinality* protocol using sets X and Y as input and output the result they receive from this protocol. Our *fuzzy join* follows the same structure as our *fuzzy cardinality* protocol, only replacing the *PSI cardinality* protocol for a *private join* protocol.

2 PRELIMINARIES

2.1 Notation

In this work, the computational and statistical security parameters are denoted by κ, λ, respectively. We use [.] notation to refer to a set. For example, $[m]$ implies the set $\{1, \ldots, m\}$. Additionally, we use $[i, j]$ to denote the set $\{i, \ldots, j\}$. We denote the concatenation of two bit strings x and y by $x||y$. The symbol $\mathbb{Z}_M$ represents the set of all elements modulo M, and $\mathbb{Z}_M^N$ is used to represent the set of all vectors of length N and components in $\mathbb{Z}_M$.

We represent a 2-out-of-2 additive sharing of x modulo M as $[\![x]\!]_M$. In this sharing scheme, the shares held by parties A and B are denoted as $[\![x]\!]_M^A \in \mathbb{Z}_M$ and $[\![x]\!]_M^B \in \mathbb{Z}_M$, respectively. These shares satisfy the relation $[\![x]\!]_M^B + [\![x]\!]_M^A = x \pmod M$.

For any set $A \subset [2^u]^d$, we define two density parameters - cell density and neighborhood density. For any grid parameter δ, we can partition the input domain $[2^u]^d$ in grid cells each of side length 2δ or into disjoint L_∞ balls each of radius δ. The cell density parameter $\rho_{c,\delta}^A$ represents the maximum number of points in A contained within any grid cell, and neighborhood density $\rho_{\eta,\delta,}^A$ to be the maximum number of points in A within any neighborhood of a grid cell. For simplicity, the δ parameter in the subscript is omitted when it is clear from the context.

We define the function $\text{cshift}_N : \mathbb{G}^N \times \mathbb{Z}_N \to \mathbb{G}^N$ for a group $\mathbb{G}$ and $N \in \mathbb{N}$ in the following way:

$$\boldsymbol{v}' = \text{cshift}_N(\boldsymbol{v}, x) \implies v'_i = v_{i+x \pmod N} \text{ for all } i \in [N].$$

Let $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{Z}_M^N$ and $s \in \mathbb{Z}_M$ for a modulo $M \in \mathbb{N}^{\geq 2}$ and vector length $N \in \mathbb{N}$. We use $\boldsymbol{v} = \boldsymbol{u} + s \pmod M$ to denote $\boldsymbol{v} = \boldsymbol{u} + \boldsymbol{1} \cdot s$, where $\boldsymbol{1} \in \mathbb{Z}_M^N$ contains the element 1 in all its components.

All protocols in this work are secure in the semi-honest model, as defined in Appendix B.1.

2.2 Fuzzy PSI Functionalities

We formally define fuzzy PSI and its related functionalities as:

Fuzzy PSI Ideal Functionalities

Parameters: Input set sizes n, m, input length ℓ, $\mathcal{D} = \mathbb{Z}_{2^u}^d$ for some $u \in \mathbb{N}$, $\texttt{dist} : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ and radius $\delta \in \mathbb{R}$, and associated length size σ for $\mathcal{F}_{\text{fuzzyJoin}}$.

Behavior:

- The two parties receiver Alice and sender Bob input sets $A, B \subset \mathcal{D}$ with $n = |A|$ and $m = |B|$
- Only for $\mathcal{F}_{\text{fuzzyJoin}}$: the sender and receiver also input associated data dictionaries AD^S, AD^R respectively with key sets A, B and values in $\{0,1\}^\ell$,
- Define outputs for each functionality as follows:
  - $\mathcal{F}_{\text{fuzzyPSI}}$: output set $\{a \in A \mid \texttt{dist}(a, b) < \delta, b \in B\}$ to the receiver Alice
  - $\mathcal{F}_{\text{fuzzyCard}}$: output $|\{a \in A \mid \texttt{dist}(a, b) < \delta, b \in B\}|$ to the receiver Alice
  - $\mathcal{F}_{\text{fuzzyJoin}}$:
    - Initialize $t = 0$.
    - For every $(a, b) \in A \times B$ where $\texttt{dist}(a, b) < \delta$:
      - sample $u_t \leftarrow_\$ \{0,1\}^{2\sigma}$
      - set v_t such that $u_t \oplus v_t = \text{AD}^S(a)||\text{AD}^R(b)$
      - $t \leftarrow t + 1$
  - Shuffle both $\vec{u}$ and $\vec{v}$ with the same random permutation
  - Output vectors $\vec{u}$ to sender and $\vec{v}$ to receiver

2.3 Oblivious Key-Value Store (OKVS)

An Oblivious Key-Value Store (OKVS) allows a sender S to encode a set of key-value pairs (k_i, v_i) into a data structure D, using uniformly random values, via the encoding function $D \leftarrow \text{OKVS.Encode}((k_i, v_i))$. A receiver R, upon receiving D, does not learn the encoded keys k_i, but can retrieve the corresponding value v when querying the store for a key k. The decoding process $v \leftarrow \text{OKVS.Decode}(D, k)$ will return v_i if $k = k_i$, and a random value otherwise. The detailed definition and its obliviousness property are provided in Appendix B.2.

2.4 Oblivious Pseudorandom Function (OPRF)

An OPRF is a two-party protocol where a sender, S, holds a key k for a PRF, and a receiver queries the function with an input x. The receiver learns only the output $F_{\text{k}}(x)$ and nothing about k. Meanwhile, S learns nothing about x. We formally define the ideal

functionality ($\mathcal{F}_{\mathsf{OPRF}}$) for the batch OPRF used in our protocol in Appendix B.3, which includes a detailed discussion of OPRF.

2.5 Oblivious Transfer (OT) and Its Variants

Oblivious Transfer (OT) [29] is a fundamental cryptographic primitive involving a sender with two input strings (m_0, m_1) and a receiver with a choice bit c. As a result of OT, the receiver obtains the string m_c without learning any information about the other string m_{1-c}, while the sender learns nothing about the choice bit c. [3] introduced the 1-out-of-N OT variant, which extends the 1-out-of-2 OT to support a message vector of length $N \in \mathbb{N}^{\geq 2}$ (i.e., the sender's input is $(m_1, \ldots, m_N)$. [17] introduces the k-out-of-N OT variant which allows the receiver choose k vectors from the N sender vectors. In [26], the concept of sparse OT was introduced, where N is exponentially larger than k.

The 1-out-of-N Shared OT (SOT) was formally defined in [28], extending the 1-out-of-N OT variant to support an additively secret-shared choice index c and to output additive secret shares of the selected message. Below, we present the 1-out-of-N SOT functionality defined in [28].

1-out-of-N SOT Ideal Functionality $\mathcal{F}_{\mathrm{SOT}_M^N}$

Parameters: Two parties Alice and Bob.
Behavior:

- Upon receiving a message $(\mathtt{choose}, [\![c]\!]_N^B)$ from Bob: Ignore any subsequent $(\mathtt{choose}, [\![c]\!]_N^B)$ messages. If $[\![c]\!]_N^B \notin \mathbb{Z}_N$, then send $(\mathtt{invalid\ input})$ to both parties and halt. Store $[\![c]\!]_N^B$ and send the public delayed message $(\mathtt{chosen})$ to Alice.
- Upon receiving a message $(\mathtt{sample\ share})$ from Alice: Ignore any subsequent messages $(\mathtt{sample\ share})$. Sample $[\![\vec{m}_c]\!]_M^A \in_R \mathbb{Z}_M$, store it internally and send it to Alice.
- Upon receiving a message $(\mathtt{propose}, [\![c]\!]_N^A, \vec{m})$ from Alice: Ignore any subsequent $(\mathtt{propose}, [\![c]\!]_N^A, \vec{m})$ messages. If it is not the case that $\vec{m} \in \mathbb{Z}_M^N$, $[\![c]\!]_N^A \in \mathbb{Z}_N$ and $[\![\vec{m}_c]\!]_M^A$ is currently stored, send $(\mathtt{invalid\ input})$ to both parties and halt. If it is the case, send $[\![\vec{m}_c]\!]_M^B = \vec{m}_c + [\![\vec{m}_c]\!]_M^A \pmod M$ to Bob.

2.6 Spatial Hashing

For domain $\mathcal{U}^d$ and grid parameter δ, the spatial hashing scheme is defined using two functions (cellhash, sphash), where each function takes as input a subset of $\mathcal{U}^d$ and outputs a dictionary. For any $x \in \mathcal{U}^d$, let $\mathsf{cell}_\delta(x) = (\lfloor x_1/2\delta \rfloor, \ldots, \lfloor x_d/2\delta \rfloor)$ and $\mathsf{trunc}_t(x) = (x \bmod 2^t)$. Here, the function cell intuitively maps any point in the domain to the unique grid cell or tile if the entire domain is tiled by d-dimensional hypercubes of side length 2δ. Let the set of all grid cells be represented by C and let $\mathsf{neigh}(x \in \mathcal{U}^d)$ output the set of all cells $\subset C$ which neighbor point x. The function trunc_t truncates the input number x to its last t bits. In our construction, we use the truncate to round number to their last $t^* = \lceil \log(6\delta) \rceil$ bits. Hence, throughout any invocation of trunc refers to trunc_{t^*}. Now we can present the spatial hashing functions for rid parameter δ:

- $D_X \leftarrow \mathsf{cellhash}(X, \rho)$: outputs a dictionary, where for each $x \in X$ dictionary D_X contains key-value pairs $(\mathsf{cell}(x), j)$ and $\mathsf{trunc}(x)$ for $j \in [\rho]$.
- $D_Y \leftarrow \mathsf{sphash}(Y)$: Outputs a dictionary D_Y which is constructed as follows:
  - Initialize dictionary D' with $D'[c] = 1$ for $c \in C$
  - For each $y \in Y$, for each $c \in \mathsf{neigh}(y)$:
    $$D_Y.\mathsf{insert}((c, D'[c]++), \mathsf{trunc}(y))$$
  This function outputs a dictionary, where for each element $y \in Y$, it inserts multiple key-value pairs, one for each neighboring cell of y.

Previous works, including [14] and [37], employ these hashing functions (or their minor modifications) for the coarse mapping of input sets X and Y in their fuzzy PSI protocols. Specifically, they prove a version of Theorem 2 (in Appendix), which we also use in this paper. The theorem essentially implies that all fuzzy matches between the two sets X and Y will have matching cell indexes in the dictionaries output by sphash and cellhash. Further, even if two non-close elements in X and Y have matching cell indexes in sphash and cellhash, then their truncated values will have a distance greater than δ as well. This second case arises when points in X and Y are in neighboring cells, but they aren't δ-close.

3 BUILDING BLOCKS

3.1 Distance-aware Random OT (da-ROT)

We present the da-ROT functionality as below, which supports fuzzy PSI and can be seamlessly applied towards "refined filtering" of [9] in a straightforward way. Specifically, the two parties execute the da-ROT protocol such that the sender obtains $\{r_0, r_1\}$ and the receiver obtains r_0 if their input points are close. The sender then obliviously transfers r_0 to the receiver using OKVS, enabling the receiver to check whether their point belongs to the fuzzy intersection.

da-ROT Ideal Functionality $\mathcal{F}_{\mathsf{da\text{-}ROT_p}}$

Parameters: Input domain $\mathcal{D} = \mathbb{Z}_v^d$ with $v \in \mathbb{N}^{\geq 2}$, output group $\mathbb{Z}_u$, threshold distance $\delta \in \mathbb{R}$, and distance metric $\mathtt{dist_p} : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$

Input: Sender S inputs $a \in \mathcal{D}$, and receiver R inputs $b \in \mathcal{D}$.

Behavior:

- Sample $r_0, r_1 \leftarrow_\$ \mathbb{Z}_u$
- Set $c = \begin{cases} 0 & \text{if } \mathtt{dist_p}(a,b) \overset{?}{\leq} \delta \\ 1 & \text{if otherwise} \end{cases}$
- Output (r_0, r_1) to sender and r_c to receiver

We designed da-ROT protocols for the distance metrics L_∞, L_1, and L_2, with the L_2 protocol specifically restricted to dimension $d = 2$. These protocols follow a common framework, differing only slightly in the formulas used in Step 2 and Step 5. Therefore, we present them as a unified protocol in Figure 2, highlighting the specific differences when applicable. Our da-ROT protocol takes advantage of the relatively small input domain (v), dimension (d), and threshold (δ) values (e.g., $v = 2^8$, $d \in 2, 6, 10$, and $\delta \in 10, 30$) when applied to our fuzzy PSI protocol. As a result, the protocol remains efficient even though its communication and computational complexity scale linearly with the size of the set $\mathbb{Z}_v^d$. Next, we

present the correctness and security proofs for our da-ROT protocol along with an explanation of the protocol. The asymptotic efficiency of the protocol is presented in Appendix C.1.

Theorem 1. *Protocol $\Pi_{da\text{-}ROT}$ securely realizes the da-ROT functionality $\mathcal{F}_{da\text{-}ROT}$ for distance metrics $\mathsf{p} \in \{\infty, 1, 2\}$ and domain $\mathcal{D} = \mathbb{Z}_v^d$, where $d \in \mathbb{N}$ and $v \in \mathbb{N}^{\geq 2}$, against a PPT semi-honest adversary in the $\mathcal{F}_{SOT}$-hybrid model.*

Proof. We begin the proof by showing that the output distribution of the functionality and the joint output distribution of the two parties in the protocol are computationally indistinguishable. To establish this, we divide the proof into three cases, corresponding to each metric $\mathsf{p} \in \{\infty, 1, 2\}$. We then proceed to show that our protocol can be simulated.

Infinity norm. When $\mathsf{p} = \infty$, the protocol defines modulo $M = d + 1$, ensuring that no wrap-around occurs during the execution of protocol operations. The protocol starts by having the two parties' non-interactively additive secret share vector $\boldsymbol{a}$ mod-v. In the second step, R and S run d instances of $\mathcal{F}_{\text{SOT}}$, where for each instance $i \in [d]$, they provide shares $[\![a_i]\!]_v$ as input and S also provides vector $\boldsymbol{m}_\infty^{(i)}$ as defined in the first part of Eq (1). As output, for each instance $i \in [d]$, the two parties receive their respective shares of $[\![h_i]\!]_M$. Based on the definition of $\mathcal{F}_{\text{SOT}}$ and how S constructs $\boldsymbol{m}_\infty^{(i)}$, we have the following equations governing the h_i values:

$$h_i = \begin{cases} 1 & \text{if } a_i \in [b_i - \delta, b_i + \delta], \\ 0 & \text{otherwise.} \end{cases}$$

In Step 3, the two parties homomorphically compute additive shares mod-M of $z = \sum_{i\in[d]} h_i$. The following equation holds based on the previously presented definition of h_i.

$$z = d \iff \forall_{i\in[d]}\ a_i \in [b_i - \delta, b_i + \delta]$$

In Steps 4-5, S samples and outputs $r \in_R \mathbb{Z}_u$ and then executes a single instance of $\mathcal{F}_{\text{SOT}}$. The two parties provide the shares $[\![z]\!]_M$ to this final $\mathcal{F}_{\text{SOT}}$ instance, and S also provides a message vector $\boldsymbol{w}_\infty$ as defined in the first part of Eq (3). As output, the parties get 2-out-of-2 additive shares mod-u of y. Based on the previously described equation for z and how S builds $\boldsymbol{w}_\infty$, we get the following equation describing y:

$$y = \begin{cases} 0, & \text{if } z = d, \\ r, & \text{otherwise.} \end{cases} = \begin{cases} 0, & \text{if } \forall_{i\in[d]}\ a_i \in [b_i - \delta, b_i + \delta], \\ r, & \text{otherwise.} \end{cases} = \begin{cases} 0, & \text{if } \mathtt{dist}_{\mathsf{p}}(\boldsymbol{a}, \boldsymbol{b}) \leq \delta, \\ r, & \text{otherwise.} \end{cases}$$

Finally, R outputs $[\![y]\!]_u^R$ and S outputs $(-[\![y]\!]_u^S, r - [\![y]\!]_u^S)$. Clearly, $[\![y]\!]_u^R = -[\![y]\!]_u^S$ if $\mathtt{dist}_{\mathsf{p}}(\boldsymbol{a}, \boldsymbol{b}) \leq \delta$, and $[\![y]\!]_u^R = r - [\![y]\!]_u^S$ otherwise. Therefore, we can conclude that the protocol and functionality have indistinguishable output distributions when $\mathsf{p} = \infty$.

L_1 norm. When $\mathsf{p} = 1$, the protocol defines modulo $M = d \cdot (\delta + 1) + 1$. Similar to the first case, after the initial two steps, the parties receive $[\![h_i]\!]_M$ from each $\mathcal{F}_{\text{SOT}}$, with h_i being described by

$$h_i = \begin{cases} |a_i - b_i|, & \text{if } |a_i - b_i| \leq \delta, \\ \delta + 1, & \text{otherwise.} \end{cases}$$

After homomorphically computing additive secret shares mod-M of $z = \sum_{i\in[d]} h_i$, we have:

$$z \leq \delta \iff \sum_{i\in[d]} |a_i - b_i| \leq \delta$$

First, suppose that $z \leq \delta$. Assuming this implies that $h_i = |a_i - b_i|$ for all $i \in [d]$, otherwise we would have $z > \delta$. In turn, this implies that $z = \sum_{i\in[d]} |a_i - b_i| \leq \delta$. Now, suppose $\sum_{i\in[d]} |a_i - b_i| \leq \delta$. Since $\sum_{i\in[d]} |a_i - b_i| \leq \delta$, it must be true that $|a_i - b_i| \leq \delta$ for all $i \in [d]$, implying that $h_i = |a_i - b_i|$ for all $i \in [d]$. Thus, we have $z = \sum_{i\in[d]} h_i = \sum_{i\in[d]} |a_i - b_i| \leq \delta$.

In Step 4-5, S samples $r \in \mathbb{Z}_u$, and then the two parties execute an $\mathcal{F}_{\text{SOT}}$ instance where S provides message vector $\boldsymbol{w}_1$ as input and the two parties provide shares $[\![z]\!]_M$ as input. As output, the parties get shares $[\![y]\!]_u$ such that y is governed by

$$y = \begin{cases} 0, & \text{if } z \leq \delta, \\ r, & \text{otherwise.} \end{cases} = \begin{cases} 0, & \text{if } \sum_{i\in[d]} |a_i - b_i| \leq \delta, \\ r, & \text{otherwise.} \end{cases} = \begin{cases} 0, & \text{if } \mathtt{dist}_1(\boldsymbol{a}, \boldsymbol{b}) \leq \delta, \\ r, & \text{otherwise.} \end{cases}$$

The remaining proof steps follow similarly to the previous case.

L_2 norm. When $\mathsf{p} = 2$, the protocol defines modulo $M = d \cdot (\delta + 1) + 1$ and sets dimension $d = 2$. The first proof steps for this case follow as in the previous two cases, with the only difference being the message vectors $\boldsymbol{m}_2^{(i)}$ provided by S to the first $d = 2$ $\mathcal{F}_{\text{SOT}}$ instances. The parties receive $[\![h_0]\!]_M$ and $[\![h_1]\!]_M$ as output, one from each $\mathcal{F}_{\text{SOT}}$ instance, where h_0 and h_1 are described by

$$h_0 = \min_{x\in\mathbb{Z}^{\geq 0}} \{(\delta + 1) - x \mid \sqrt{x^2 + (a_0 - b_0)^2} > \delta\}$$

$$h_1 = \begin{cases} |a_1 - b_1| & \text{if } |a_1 - b_1| \leq \delta, \\ \delta + 1 & \text{otherwise.} \end{cases}$$

The value $z = h_0 + h_1$ satisfies the following.

$$z \leq \delta \iff \sqrt{(a_1 - b_1)^2 + (a_0 - b_0)^2} \leq \delta$$

First, suppose $\sqrt{(a_1 - b_1)^2 + (a_0 - b_0)^2} \leq \delta$. This supposition implies that $h_1 = |a_1 - b_1|$, and

$$\begin{aligned} & h_1 < \min_{x\in\mathbb{Z}^{\geq 0}} \{x \mid \sqrt{x^2 + (a_0 - b_0)^2} > \delta\} \\ \Leftrightarrow\ & h_1 - \min_{x\in\mathbb{Z}^{\geq 0}} \{x \mid \sqrt{x^2 + (a_0 - b_0)^2} > \delta\} < 0 \\ \Leftrightarrow\ & h_1 + (\delta + 1) - \min_{x\in\mathbb{Z}^{\geq 0}} \{x \mid \sqrt{x^2 + (a_0 - b_0)^2} > \delta\} < \delta + 1 \\ \Leftrightarrow\ & h_1 + \min_{x\in\mathbb{Z}^{\geq 0}} \{(\delta + 1) - x \mid \sqrt{x^2 + (a_0 - b_0)^2} > \delta\} < \delta + 1 \\ \Leftrightarrow\ & h_1 + h_0 < \delta + 1 \\ \Leftrightarrow\ & h_1 + h_0 \leq \delta \end{aligned}$$

Now, suppose $\sqrt{(a_1 - b_1)^2 + (a_0 - b_0)^2} > \delta$. This supposition implies the following:

Parameters:

- $\mathrm{p} \in \{\infty, 1, 2\}$; Radius $\delta \in \mathbb{N}$; Domain $\mathcal{D} = \mathbb{Z}_v^d$ with $v \in \mathbb{N}^{\geq 2}$; Group $\mathbb{Z}_u$.
- Dimension $d = 2$ if $\mathrm{p} = 2$, and $d \in \mathbb{N}$ otherwise.
- The SOT functionality as described in Section 2.5.

Inputs: Receiver R and sender S input points $\boldsymbol{a} \in \mathbb{Z}_v^d$ and $\boldsymbol{b} \in \mathbb{Z}_v^d$, respectively.

Protocol: Let $M = d + 1$ if $\mathrm{p} = \infty$, and $M = d \cdot (\delta + 1) + 1$ otherwise. For all the instances of SOT executed in this protocol, S and R play the roles of sender and receiver, respectively.

(1) For each $i \in [d]$, R set $[\![a_i]\!]_v^A = a_i$ and S sets $[\![a_i]\!]_v^B = 0$.

(2) Both parties invoke a 1-oo-v SOT for every $i \in [d]$, where they input $[\![a_i]\!]_v$, and S also inputs the message vector $\boldsymbol{m}_{\mathrm{p}}^{(i)} \in \mathbb{Z}_M^v$, which is computed using Eq (1) or Eq (2), depending on the metric p. As a result, each party obtains $[\![h_i]\!]_M$ as output.

$$m_{\infty,j}^{(i)} = \begin{cases} 1 & \text{if } j \in [b_i - \delta, b_i + \delta], \\ 0 & \text{otherwise.} \end{cases} \quad \text{and} \quad m_{1,j}^{(i)} = \begin{cases} |j - b_i| & \text{if } |j - b_i| \leq \delta, \\ \delta + 1 & \text{otherwise.} \end{cases} \tag{1}$$

$$m_{2,j}^{(0)} = \min_{x \in \mathbb{Z}^{\geq 0}} \{(\delta + 1) - x \mid \sqrt{x^2 + (j - b_0)^2} > \delta\} \quad \text{and} \quad m_{2,j}^{(1)} = \begin{cases} |j - b_1| & \text{if } |j - b_1| \leq \delta, \\ \delta + 1 & \text{otherwise.} \end{cases} \tag{2}$$

(3) The parties compute $[\![z]\!]_M = \sum_{i \in [d]} [\![h_i]\!]_M$.

(4) S samples and outputs $r \in_R \mathbb{Z}_u$.

(5) Both parties execute a single 1-oo-M SOT for which they input $[\![z]\!]_M$, and S also inputs the message vector $\boldsymbol{w}_{\mathrm{p}} \in \mathbb{Z}_u^M$, which is computed using Eq (3), depending on the metric p. As a result, the parties obtain $[\![y]\!]_u$ as output.

$$w_{\infty,j} = \begin{cases} 0 & \text{if } j = d, \\ r & \text{otherwise.} \end{cases} \quad \text{and} \quad w_{1,j} = \begin{cases} 0 & \text{if } j \leq \delta, \\ r & \text{otherwise.} \end{cases} \quad \text{and} \quad w_{2,j} = \begin{cases} 0 & \text{if } j \leq \delta, \\ r & \text{otherwise.} \end{cases} \tag{3}$$

(6) R outputs $[\![y]\!]_u^R$ and S outputs $(-[\![y]\!]_u^S, r - [\![y]\!]_u^S)$.

Figure 2: Our da-ROT Protocol $\Pi_{\text{da-ROT}_{\mathrm{p} \in \{\infty,1,2\}}}$

$$\begin{aligned} & h_1 \geq \min_{x \in \mathbb{Z}^{\geq 0}} \{x \mid \sqrt{x^2 + (a_0 - b_0)^2} > \delta\} \\ \Leftrightarrow \quad & h_1 + (\delta + 1) - \min_{x \in \mathbb{Z}^{\geq 0}} \{x \mid \sqrt{x^2 + (a_0 - b_0)^2} > \delta\} \geq \delta + 1 \\ \Leftrightarrow \quad & h_1 + h_0 > \delta \end{aligned}$$

Proving that indeed $z \leq \delta \iff \sqrt{(a_1 - b_1)^2 + (a_0 - b_0)^2} \leq \delta$. The remaining proof steps for this case follow as for case $\mathrm{p} = 1$.

Based on the proofs for these three individual cases, we conclude that the output distribution of this protocol is indistinguishable from the functionality's output distribution for all $\mathrm{p} \in \{\infty, 1, 2\}$. We now proceed to describe the simulators. For this part of the proof, we omit detailed discussion for each distance metric and instead provide a sketch of the simulators, as the simulation process is essentially trivial.

Corrupt Receiver. By inspecting the protocol, it is straightforward to see that both sender S and receiver R only receive messages from $\mathcal{F}_{\text{SOT}}$ instances, where these messages are the output 2-out-of-2 secret additive secret shares. We simulate the output for the SOT instances executed at step 2 by uniformly sampling elements mod-M, since these SOT instances output shares mod-M. This leaves us the last SOT executed at step 5. For this last SOT, we output r_c as the output share, where r_c is the receiver's output sent by the functionality $\mathcal{F}_{\text{da-ROT}}$.

Corrupt Sender. The simulation proceeds similarly for the sender. We simulate the output for the SOT instances executed at step 2 by uniformly sampling elements mod-M, and then simulate the final SOT instance's output at step 5 using the sender's functionality output. Let (r_0, r_1) be the sender's output received from $\mathcal{F}_{\text{da-ROT}}$. For the last SOT instance, we output $-r_0 \pmod u$ as the output share.

□

3.2 Sparse SOT

This section introduces the new ideal functionality called Sparse SOT ($\mathcal{F}_{\text{SpSOT}}$), which generalizes the 1-out-of-N Sparse Oblivious Transfer (SOT) framework. This primitive builds upon the concepts proposed in [13, 26, 28].

The $\mathcal{F}_{\text{SpSOT}}$ functionality involves two parties. The receiver, R, inputs a sparse index set $I^R \subseteq \mathcal{D}$ of size n^R and an additive share of the choice index, $[\![c^{(i)}]\!]_N^R$, for each index in I^R. The sender, S, inputs its own sparse index set $I^S \subseteq \mathcal{D}$ of size n^S, along with an additive share of the choice index, $[\![c^{(i)}]\!]_N^S$, and an associated message vector, $\mathbf{m}^{(i)} \in \mathbb{Z}_M^N$. Upon completing the Sparse SOT functionality, the receiver R and the sender S will each obtain n^R and n^S additive secret shares, denoted as $\boldsymbol{y}^R$ and $\boldsymbol{y}^S$, respectively. Each share is associated with a sparse index within their respective sets, I^R and I^S. The relationships governing these secret share values are outlined in the equations below.

$$\boldsymbol{y}^R \in_R \mathbb{Z}_M^{n^R}, \boldsymbol{y}^S \in_R \mathbb{Z}_M^{n^S}$$

$y_j^R + y_k^S = m_{c^{(i)}}^{(i)}$, for every $i_j^R \in I^R$ and $i_k^S \in I^S$, such that $i = i_j^R = i_k^S$.

In other words, if both parties share a common index i in their sparse index sets, they execute a 1-out-of-N SOT using their respective choice index shares and the sender's message vector as inputs. Otherwise, the output share associated with i is a uniformly random element sampled from $\mathbb{Z}_M$. We formally define the Sparse SOT ideal functionality as follows.

Sparse SOT Ideal Functionality $\mathcal{F}_{\text{SpSOT}}$

Parameters:

- Sparse index domain $\mathcal{I}$.
- Moduli $N, M \in \mathbb{N}^{\geq 2}$.
- Input set sizes $n^R, n^S \in \mathbb{N}$.

Inputs:

- R: An ordered set $I^R = \{i_0^R, \ldots, i_{n^R-1}^R\} \subseteq \mathcal{I}$ of size n^R and a vector additive secret shares $(\llbracket c^{(i)} \rrbracket_N^R)_{i \in I_R}$.
- S: An ordered set $I^S = \{i_0^S, \ldots, i_{n^S-1}^S\} \subseteq \mathcal{I}$ of size n^S and a vector of pairs of additive secret shares and message vectors $(\llbracket c^{(i)} \rrbracket_N^S, \boldsymbol{m}^{(i)})_{i \in I_S}$, where $\boldsymbol{m}^{(i)} \in \mathbb{Z}_M^N$.

Behavior:

- Let $\boldsymbol{y}^R \in \mathbb{Z}_M^{n^R}$ and $\boldsymbol{y}^S \in \mathbb{Z}_M^{n^S}$.
- For every $i \in I^R \cap I^S$:
  - Reconstruct $c^{(i)}$ using $\llbracket c^{(i)} \rrbracket_N^R$ and $\llbracket c^{(i)} \rrbracket_N^S$
  - Sample 2-out-of-2 additive secret shares $(\llbracket m_{c^{(i)}}^{(i)} \rrbracket_M^S, \llbracket m_{c^{(i)}}^{(i)} \rrbracket_M^R)$ of $m_{c^{(i)}}^{(i)}$.
  - Set $y_j^R \leftarrow \llbracket m_{c^{(i)}}^{(i)} \rrbracket_M^R$, where $i = i_j^R$.
  - Set $y_j^S \leftarrow \llbracket m_{c^{(i)}}^{(i)} \rrbracket_M^S$, where $i = i_j^S$.
- For every $i \in I^S \setminus I^R$:
  - Sample $y_j^S \in_R \mathbb{Z}_M$, where $i = i_j^S$.
- For every $i \in I^R \setminus I^S$:
  - Sample $y_j^R \in_R \mathbb{Z}_M$, where $i = i_j^R$.
- Send $\boldsymbol{y}^R$ and $\boldsymbol{y}^S$ to R and S, respectively.

To construct a protocol for $\mathcal{F}_{\text{SpSOT}}$, we leverage concepts from [28] and the well-known reduction from ROT to OT. These are combined with the OPRF and the OKVS scheme to satisfy the necessary sparsity requirements. Figure 3 presents our sparse OT protocol.

A good starting point for understanding our Sparse SOT protocol is to examine how the sender S builds the OKVS D, which is sent to the receiver R at protocol step 6. Specifically, party S encodes the following key-value pair set E into the OKVS D:

$$E = \{((i,j), w_j^{(i)}) \mid (i,j) \in I^S \times \mathbb{Z}_N\}$$
$$w_j^{(i)} \leftarrow u_j^{(i)} - \llbracket z^{(i)} \rrbracket_M^S - F_\theta(i,j) - h^{(i)}$$

After receiving D, party R evaluates the OKVS to obtain $q_{\llbracket c^{(i)} \rrbracket_N^R}^{(i)} \leftarrow \text{OKVS.Decode}(D, (i, \llbracket c^{(i)} \rrbracket_N^R))$. It then outputs shares $\llbracket z^{(i)} \rrbracket_M^R$ as:

$$\llbracket z^{(i)} \rrbracket_M^R \leftarrow q_{\llbracket c^{(i)} \rrbracket_N^R}^{(i)} + f^{(i)} + H_\phi(i) \text{ for every } i \in I^R$$

This implies that the following relationship will hold for every $i \in I^S \cap I^R$, based on the definition of $f^{(i)}$ and $h^{(i)}$.

$$\llbracket z^{(i)} \rrbracket_M^R = u_{\llbracket c^{(i)} \rrbracket_N^R}^{(i)} - \llbracket z^{(i)} \rrbracket_M^S \text{ for every } i \in I^R$$

By the definition of cshift provided in Section 2.1 and its application in protocol step 4, we have:

$$\llbracket z^{(i)} \rrbracket_M^R = m_{c^{(i)}}^{(i)} - \llbracket z^{(i)} \rrbracket_M^S \text{ for every } i \in I^R$$

Since $\llbracket z^{(i)} \rrbracket_M^S$ is uniformly sampled by S at Step 3, and S outputs $\llbracket z^{(i)} \rrbracket_M^S$ as its own share, both parties receive additive shares mod-M of $m_{c^{(i)}}^{(i)}, \forall i \in I^R \cap I^S$. Note that R only learns $\llbracket z^{(i)} \rrbracket_M^R$, because in the first protocol step, R obliviously queried $F_\theta(i,j)$ only for $j = \llbracket c^{(i)} \rrbracket_M^R$, and learning any other information about $\boldsymbol{m}^{(i)}$ would require querying F_θ for different j values.

Next, we consider the case where $i \in I^S \setminus I^R$. We show that S outputs $\llbracket z^{(i)} \rrbracket_M^S \in_R \mathbb{Z}_M$ such that R does not learn anything about it. The fact that $\llbracket z^{(i)} \rrbracket_M^S \in_R \mathbb{Z}_M$ is uniformly sampled from $\mathbb{Z}_M$ is evident from protocol step 3. Now, party S does send the vector $\boldsymbol{w}^{(i)} \in \mathbb{Z}_M^N$ as part of OKVS D to R, where

$$w_j^{(i)} = u_j^{(i)} - \llbracket z^{(i)} \rrbracket_M^S - F_\theta(i,j) - h^{(i)}.$$

However, the receiver does not hold $F_\theta(i,j)$ for any j, since $i \in I^S \cap I^R$. This implies that the receiver does not learn anything about $\llbracket z^{(i)} \rrbracket_M^S$.

Now, we consider the final case where $i \in I^R \setminus I^S$. We know that R computes its shares $\llbracket z^{(i)} \rrbracket_M^R$ according to the following equation:

$$\llbracket z^{(i)} \rrbracket_M^R \leftarrow q_{\llbracket c^{(i)} \rrbracket_N^R}^{(i)} + f^{(i)} + H_\phi(i)$$

Because $H_\phi(i)$ is pseudorandom and unknown to S, and $f^{(i)}$ is pseudorandom to R, we can conclude that $\llbracket z^{(i)} \rrbracket_M^R$ is indistinguishable from an element uniformly sampled from $\mathbb{Z}_M$ and S does not learn any information about it.

We formally define the security of our Sparse SOT protocol (Π_{SpSOT}) in Theorem 3 and provide the proof in Appendix D.1. At a high level, the security of our protocol follows from the security of OKVS and the way we use two $\mathcal{F}_{\text{OPRF}}$ instances, where the two parties reverse roles between the two instances. The asymptotic efficiency of Π_{SpSOT} is detailed in Appendix C.2.

4 FUZZY PSI FRAMEWORK

The key building block of our PSI framework is the $\mathcal{F}_{\text{Sp-daROT}}$ functionality, introduced in the next subsection. This functionality can be seen as a sparse variant of the $\mathcal{F}_{\text{da-ROT}}$ functionality. In Section 4.1, we also present a black-box compiler that constructs a protocol for $\mathcal{F}_{\text{Sp-daROT}}$ using $\mathcal{F}_{\text{da-ROT}}$ and $\mathcal{F}_{\text{SpSOT}}$. Subsequently, in Section 4.2, we introduce our Fuzzy PSI framework, which supports L_∞ and L_p ($\text{p} \in \mathbb{N}$) norms. The framework can be easily adapted to support other fuzzy PSI-related functionalities.

4.1 Sparse distance-aware random OT (Sp-daROT) Functionality and Its Compiler

Functionality $\mathcal{F}_{\text{Sp-daROT}}^{\mathcal{D},u,\texttt{dist},\delta}$ with parameters input domain $\mathcal{D}$, output group $\mathbb{Z}_u$, distance metric $\texttt{dist}$, threshold $\delta(>0)$ defines the sparse variant of the $\mathcal{F}_{\text{da-ROT}}^{\mathcal{D},u,\texttt{dist},\delta}$ functionality in a natural way - where each party has multiple da-ROT inputs ($\in \mathcal{D}$), each with a corresponding index ($\in \mathcal{I}$). Hence, each party inputs a dictionary

Parameters:

- Set size n^S and n^R
- Two PRFs $F\colon \mathcal{I} \times \mathbb{Z}_N \to \mathbb{Z}_M$, and $H\colon \mathcal{I} \to \mathbb{Z}_M$
- The OPRF functionality, OKVS scheme, and cshift function described Section 2.4, Section 2.3, and Section 2.1, respectively.

Inputs:

- The S's input: An ordered set $I^S = \{i^S_0, \ldots, i^S_{n^S-1}\} \subseteq \mathcal{I}$ of size n^S and a vector of pairs of additive secret shares and message vectors $(\llbracket c^{(i)} \rrbracket^S_N, \boldsymbol{m}^{(i)})_{i \in I_S}$, where $\boldsymbol{m}^{(i)} \in \mathbb{Z}^N_M$.
- The R's input: An ordered set $I^R = \{i^R_0, \ldots, i^R_{n^R-1}\} \subseteq \mathcal{I}$ of size n^R and a vector additive secret shares $(\llbracket c^{(i)} \rrbracket^R_N)_{i \in I_R}$.

Protocol: Assume every arithmetic operation is done modulo M.

(1) S and R run $\mathcal{F}_{\mathsf{OPRF}}$ for PRF F, with R querying points $(i, \llbracket c^{(i)} \rrbracket^R_N)_{i \in I^R}$. As output, S receives an PRF key θ, and R receives vector $\boldsymbol{f} = (F_\theta(i, \llbracket c^{(i)} \rrbracket^R_N))_{i \in I^R}$.
Let $f^{(i)} = f_j$, where j is the position of vector $\boldsymbol{f}$ for which $f_j = F_\theta(i, \llbracket c^{(i)} \rrbracket^R_N))$.
(2) S and R run $\mathcal{F}_{\mathsf{OPRF}}$ for PRF H, with S querying points I^S. As output, R receives key ϕ, and S receives vector $\boldsymbol{h} = (H_\phi(i))_{i \in I^S}$.
Let $h^{(i)} = h_j$, where j is the position of vector $\boldsymbol{h}$ for which $h_j = H_\phi(i)$.
(3) S samples $\llbracket z^{(i)} \rrbracket^S_M \in_R \mathbb{Z}_M$, for every $i \in I^S$.
(4) S computes $\boldsymbol{u}^{(i)} \leftarrow \mathsf{cshift}_N(\boldsymbol{m}^{(i)}, \llbracket c^{(i)} \rrbracket^S_N)$ for every $i \in I^S$.
(5) S computes $w^{(i)}_j \leftarrow u^{(i)}_j - \llbracket z^{(i)} \rrbracket^S_M - F_\theta(i, j) - h^{(i)}$, for every pair $(i, j) \in I^S \times \mathbb{Z}_N$.
(6) S sends $D \leftarrow \mathsf{OKVS.Encode}(E)$ to R, where $E = \{((i, j), w^{(i)}_j) \mid (i, j) \in I^S \times \mathbb{Z}_N\}$.
(7) R computes $q^{(i)}_{\llbracket c^{(i)} \rrbracket^R_N} \leftarrow \mathsf{OKVS.Decode}(D, (i, \llbracket c^{(i)} \rrbracket^R_N))$ for every $i \in I^R$.
(8) R computes $\llbracket z^{(i)} \rrbracket^R_M \leftarrow q^{(i)}_{\llbracket c^{(i)} \rrbracket^R_N} + f^{(i)} + H_\phi(i)$ for every $i \in I^R$.
(9) R and S output $\boldsymbol{y}^R$ and $\boldsymbol{y}^S$, respectively, where $\boldsymbol{y}^R = (\llbracket z^{(i^R_0)} \rrbracket^R_M, \ldots, \llbracket z^{(i^R_{n^R-1})} \rrbracket^R_M)$ and $\boldsymbol{y}^S = (\llbracket z^{(i^S_0)} \rrbracket^S_M, \ldots, \llbracket z^{(i^S_{n^S-1})} \rrbracket^S_M)$.

Figure 3: Our Sparse SOT Protocol Π_{SpSOT}

$\in \mathcal{I} \times \mathcal{D}$ of some public size. The functionality outputs two dictionaries (same size as respective inputs), for each party. The Sp-daROT sender's output dictionary and receiver's output dictionary have elements as defined by the $\mathcal{F}_{\mathsf{da\text{-}ROT}}$ functionality for each index present in both input parties' dictionaries. More formally, if the sender and receiver have inputs (i, x) and (i, y) in their input dictionaries, respectively, then their output dictionaries have elements $(i, (r_0, r_1))$ and (i, r_c) respectively where $c = [\mathtt{dist}(x, y) \overset{?}{\leq} \delta]$.

For non-intersecting indexes, the parties receive random values in the output dictionary - ensuring no party can learn the intersecting indexes given just their outputs of Sp-daROT functionality. The formal description is provided below.

Sparse Distance-Aware Random OT $\mathcal{F}^{\mathcal{D},u,\mathtt{dist},\delta}_{\mathsf{Sp\text{-}daROT}}$

Parameters: Input domain $\mathcal{D}$, index set $\mathcal{I}$, output group $\mathbb{Z}_u$, threshold distance $\delta \in \mathbb{R}$, distance metric $\mathtt{dist} : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$, sender and receiver set sizes n_0 and n_1 respectively

Input: Sender S and Receiver R input dictionaries $X \in \mathcal{I} \times \mathcal{D}$, $Y \in \mathcal{I} \times \mathcal{D}$ of sizes n_0 and n_1, respectively.

Behavior:

- Initialize empty dictionaries $X_{\mathsf{out}}, Y_{\mathsf{out}}$
- For each $(i, x) \in X$:
  - Sample $r_0, r_1 \leftarrow_\$ \mathbb{Z}_u$
  - $X_{\mathsf{out}}.\mathsf{insert}(i, (r_0, r_1))$
- For each $(i, y) \in Y$:
  - **If** $X[i] \overset{?}{=} \perp$: $Y_{\mathsf{out}}.\mathsf{insert}(i, r)$ where $r \leftarrow_\$ \mathbb{Z}_u$
  - **Else**
    - Compute $c = \begin{cases} 0 & \text{if } \mathtt{dist}(X[i], y) \overset{?}{\leq} \delta \\ 1 & \text{if otherwise} \end{cases}$
    - $Y_{\mathsf{out}}.\mathsf{insert}(i, r_c)$
- Output X_{out} to Sender S and Y_{out} to Receiver R.

0-round SOT Hybrid Protocols. Our design of the Sp-daROT compiler builds on the observation that all the da-ROT protocols proposed in Subsection 3.1 operate in the shared OT hybrid model. Moreover, in these constructions, the sender and receiver interact solely through a constant number of calls to the SOT functionality with fixed parameters. This property is formalized in the following definition.

Definition 4.1 (0-round protocol in SOT hybrid model)**.** A two-party protocol $\Pi = (\mathsf{SOTInp}, \mathsf{UpdState}, \mathsf{Out}, \mathsf{param}, k \in \mathbb{N})$ is termed a 0-round protocol in $\mathcal{F}_{\mathsf{SOT}}$ hybrid model if it has the following form:

Inputs: Sender S inputs x, and Receiver R inputs y.

Behavior:

- R and S initialize states $\sigma_R \leftarrow \{x\}$, $\sigma_S \leftarrow \{y\}$, respectively

- For $j \in [k]$:
  - R executes $i_R \leftarrow \mathsf{SOTInp}(R, \sigma_R, j)$; S executes $i_S \leftarrow \mathsf{SOTInp}(S, \sigma_S, j)$
  - R acts as receiver and S as sender, and they execute $(o_R, o_S) \leftarrow \mathcal{F}_{\mathsf{SOT}}^{\mathsf{param}[j]}(i_R, i_S)$
  - R executes $\sigma_R \leftarrow \mathsf{UpdState}(R, \sigma_R, j, o_R)$; S executes $\sigma_S \leftarrow \mathsf{UpdState}(S, \sigma_S, j, o_S)$
- R outputs $\mathsf{Out}(R, \sigma_R)$ and S outputs $\mathsf{Out}(S, \sigma_S)$

We define these 0-round protocols using three algorithms SOTInp, UpdState and Out, a parameter vector param and an integer k representing the total number of iterations or SOT calls in the protocol. In these protocols, the sender and receiver maintain a local state σ_S and σ_R, which is updated after each SOT call. The protocol has the following structure: for iteration $j \in [k]$: each party runs the SOTInp algorithm to generate the SOT input for j^{th} iteration. The output of this algorithm is input to $\mathcal{F}_{\mathsf{SOT}}$ with parameters $\mathsf{param}[j]$. The output of this SOT call is further used to update the local state of each party using the UpdState function. At the end of the k iterations, each party can use the Out function to generate the output of their protocol.

For each of our da-ROT protocols in Section 3.1, we do not explicitly present them in the aforementioned form; however, they clearly adhere to this structure, i.e., in all our protocols, the sender and receiver only invoke the SOT primitive a constant number of times, and they do not interact other than through these SOT calls.

A Sparse Compiler for 0-round SOT da-ROT Protocols. In Figure 4, we present a compiler that, given a 0-round da-ROT protocol in the $\mathcal{F}_{\mathsf{SOT}}$ hybrid model, can generate a corresponding Sp-daROT protocol in SpSOT hybrid model. The key idea behind the design of this sparse compiler is to run multiple parallel instances of da-ROT protocols (proportional to the input set size), where each set of parallel invocations of $\mathcal{F}_{\mathsf{SOT}}$ is replaced by a single call to $\mathcal{F}_{\mathsf{SpSOT}}$. More formally, in the j^{th} iteration, let the sender run $o^i \leftarrow \mathsf{SOTInp}(S, \sigma_S^i, j)$ algorithm on state σ_S^i for each index i in its input set. Then the sender input in the corresponding iteration of $\mathcal{F}_{\mathsf{SpSOT}}$ primitive call in the compiler is $\{(i, o^i)|$ for index i in input set $X\}$. The security of this compiler for Sp-daROT follows directly from the underlying da-ROT protocol. A formal security theorem, along with a proof sketch, is in Appendix D.2, with asymptotic efficiency detailed in Appendix C.3.

Parameters: Given a da-ROT protocol $\Pi = (\mathsf{SOTInp}, \mathsf{UpdState}, \mathsf{Out}, \mathsf{param}, k \in \mathbb{N})$

Inputs: Sender S inputs X and Receiver R inputs Y.
Behavior:

- S initializes states $\sigma_S^i \leftarrow \{x\}$ for each $(i, x) \in X$
- R initializes states $\sigma_R^i \leftarrow \{y\}$ for each $(i, y) \in Y$
- For $j \in [k]$:
  - R computes $X' \leftarrow \{(i, \mathsf{SOTInp}(R, \sigma_R^i, j)) \mid \text{where } (i, x) \in X\}$
  - S computes $Y' \leftarrow \{(i, \mathsf{SOTInp}(S, \sigma_S^i, j)) \mid \text{where } (i, y) \in Y\}$
  - R acts as receiver and S as sender, and they execute $(O_R, O_S) \leftarrow \mathcal{F}_{\mathsf{SpSOT}}^{\mathsf{param}[j]}(X', Y')$
  - For each $(i, x) \in X$: R updates state $\sigma_R^i \leftarrow \mathsf{UpdState}(R, \sigma_R^i, j, O_R[i])$
  - For each $(i, y) \in Y$: S updates state $\sigma_S^i \leftarrow \mathsf{UpdState}(S, \sigma_S^i, j, O_S[i])$
- R outputs $Y_{\mathsf{out}} = \{(i, \mathsf{Out}(R, \sigma_R^i)) \mid \text{where } (i, x) \in X\}$
- S outputs $X_{\mathsf{out}} = \{(i, \mathsf{Out}(S, \sigma_S^i)) \mid \text{where } (i, y) \in Y\}$

Figure 4: sparse-comp: Compiler for Sp-daROT

4.2 Our Fuzzy PSI Framework

Figure 5 illustrates our fuzzy PSI framework, which combines spatial hashing with sparse distance-aware OT (Sp-daROT). While the da-ROT protocols in Section 3.1 are restricted to specific norms and dimensions, this framework generalizes to any dimension d and any norm L_p ($p \in \mathbb{N} \cup \infty$). Its flexibility makes it a valuable foundation for future fuzzy PSI protocol designs.

High-level Outline of Fuzzy PSI Framework. Alice (the receiver) and Bob (the sender) input sets A and B, respectively, such that the cell density of A with grid parameter δ is 1.

In the protocol, Alice and Bob first run local algorithms $A' \leftarrow \mathsf{sphash}(A)$ and $\mathsf{cellhash}(B)$ respectively, where the spatial hashing is done with grid parameter δ. By correctness of spatial hashing, for each $a \in A$ and $b \in B$ that are in the same grid cells or neighboring ones, there exists some index i such that $(i, \mathsf{trunc}(a)) \in A'$ and $(i, \mathsf{trunc}(b)) \in B'$. Here, the trunc function outputs only the last $t^* = \lceil \log(6\delta) \rceil$ input bits (as discussed in Subsection 2.6) - reducing the problem of fuzzy PSI over exponentially large domain $\lceil 2^u \rceil^d$ to sparse matching over smaller domain of size $2^{dt^*} = O(\delta^d)$. Alice and Bob now execute the $\mathcal{F}_{\mathsf{Sp\text{-}daROT}}$ functionality as receiver and sender, respectively, on dictionaries A' and B', and they receive as output dictionaries A_{out} and B_{out}. By correctness of Sp-daROT, for each $a \in A, b \in B$ that are δ-close in the underlying metric space $\mathtt{dist}$, there exist (i, r_0) and $(i, (r_0, r_1))$ in the output sets A_{out} and B_{out} where i was the index of cell containing both a and b.

To complete the fuzzy PSI protocol, we want Alice to learn the elements from Bob that are δ-close to its input set A. Specifically, Alice should learn the elements from Bob that share the same "r-value" output by both dictionaries A_{out} and B_{out} for corresponding indexes. To achieve this, Bob first encrypts each of his points b using a one-time pad authenticated encryption with key r_0, where $(i, \mathsf{trunc}(b)) \in B'$ and $(i, (r_0, r_1)) \in B_{\mathsf{out}}$, and adds the encrypted points to an OKVS D, which he then sends to Alice.

Upon receiving D, Alice performs the following steps: for each element $(i, r_0) \in A_{\mathsf{out}}$, Alice checks if $\mathsf{Dec}(r_0, \mathsf{OKVS.Decode}(D, i))$ does not result in an error. If the decryption is successful, Alice adds the corresponding element to the output set Z. Due to the obliviousness property of the OKVS, Alice cannot learn any key-value pairs that were added by Bob to OKVS D, except for those that are revealed through the output set Z.

Extending to Other Fuzzy Variants. Another advantage of our modular framework is its flexibility in adapting to other fuzzy PSI functionalities, such as $\mathcal{F}_{\mathsf{fuzzyCard}}$ and $\mathcal{F}_{\mathsf{fuzzyJoin}}$, as depicted in

Parameters:

- Input domain $\mathcal{D} = \mathcal{U}^d$, threshold distance $\delta \in \mathbb{R}$, distance metric $\texttt{dist} : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ for norm $p \in [0, \infty]$, sender and receiver set sizes n and m respectively
- (cellhash, sphash) is a spatial hashing construction for domain $\mathcal{U}^d$ and grid parameter δ
- (OKVS.Encode, OKVS.Decode) be an OKVS scheme
- (Enc, Dec) be a one-time pad authenticated encryption scheme
- The functionalities for traditional PSI Cardinality, $\mathcal{F}_{\mathsf{PSICard}}$, and traditional PSI Join, $\mathcal{F}_{\mathsf{PSIJoin}}$, as described in Appendix E.
- The function trunc_t truncates the input number x to its last t bits, where $t = \lceil \log(6\delta) \rceil$ bits.

Inputs:

- The receiver Alice inputs a set $A \subset \mathcal{U}^d$ of size n. The sender Bob inputs a set B $\subset \mathcal{U}^d$ of size m with $\rho_c^B = 1$
- For $\mathcal{F}_{\mathsf{fuzzyJoin}}$ functionality, Alice and Bob have additional input dictionaries AD^R and AD^S

Protocol:

(1) Alice computes $A' \leftarrow \mathsf{sphash}(A)$. Bob computes $B' \leftarrow \mathsf{cellhash}(B, \rho_\eta^A)$

(2) The parties invoke $\mathcal{F}_{\text{Sp-daROT}}^{\mathcal{D},u,\texttt{dist},\delta}$ protocol where:
- Alice acts as receiver with input A'; and Bob acts as sender with input B'
- Alice and Bob receive outputs A_{out} and B_{out}, respectively

For $\mathcal{F}_{\mathsf{fuzzyPSI}}$ functionality:

(3) Bob initializes an empty dictionary Y, which is updated as follows:
- For each $(i, (r_0, r_1)) \in B_{\text{out}}$:
  Find $b \in B$ such that $(i, b) \in B'$
  $Y.\mathsf{insert}(i, \mathsf{Enc}(r_0, b))$

(4) Bob sends $D \leftarrow \mathsf{OKVS.Encode}(Y)$

(5) Alice initializes the output set $Z = \{\}$. For $\forall (i, r_0) \in A_{\text{out}}$:
- Compute ciphertext $c \leftarrow \mathsf{OKVS.Decode}(D, i)$
- If $b \leftarrow \mathsf{Dec}(r_0, c) \neq \perp$: $Z \leftarrow Z \cup \{b\}$

For $\mathcal{F}_{\mathsf{fuzzyCard}}$ functionality:

(6) Bob computes $Y \leftarrow \{r_0 | (i, (r_0, r_1)) \in B_{\text{out}}\}$; and Alice computes $X \leftarrow \{r_0 | (i, r_0)\} \in A_{\text{out}}\}$

(7) Parties invoke $\mathcal{F}_{\mathsf{PSICard}}$ where Alice acts as receiver, and Bob acts as sender with inputs sets X and Y respectively; the output from this functionality is Alice's output

For $\mathcal{F}_{\mathsf{fuzzyJoin}}$ functionality:

(8) Bob computes $Y \leftarrow \{(r_0, \mathsf{AD}^S(b) | (i, (r_0, r_1)) \in B_{\text{out}}, (i, \mathsf{trunc}(b)) \in B'\}$

(9) Alice computes $X \leftarrow \{(r_0, \mathsf{AD}^S(a) | (i, r_0) \in A_{\text{out}}, (i, \mathsf{trunc}(a)) \in A'\}$

(10) Parties invoke $\mathcal{F}_{\mathsf{PSIJoin}}$ where Alice acts as the receiver and Bob acts as the sender with inputs sets X and Y respectively; the output from this functionality is Alice's output

Figure 5: Our Fuzzy PSI Framework

Figure 5. As discussed earlier, for each $a \in A$ and $b \in B$ that are δ-close in the underlying metric space dist, there exist corresponding (i, r_0) and $(i, (r_0, r_1))$ in the output sets A_{out} and B_{out} for some index i. Therefore, for the fuzzyCard functionality, the parties can simply check how many r_0 values in the two sets A_{out} and B_{out} match. This matching can be efficiently performed using a traditional PSI cardinality protocol, as outlined in Step 7 of Figure 5.

Similarly, we show in Steps 8-10 how the $\mathcal{F}_{\mathsf{fuzzyJoin}}$ can be reduced to the traditional join functionality $\mathcal{F}_{\mathsf{PSIJoin}}$. In this case, the approach is to perform the standard join operation on the matching r_0 values from the sets A_{out} and B_{out}.

Theorem 5, which formalizes our fuzzy PSI protocol, is stated in Appendix D.3, where its correctness and security are also formally established. We also get Theorem 6 for other fuzzy PSI related functionalities in Appendix D.3. The asymptotic efficiency of the protocols is detailed in Appendix C.4.

5 PERFORMANCE EVALUATION

We provide experimental details and compare the performance of our fuzzy PSI protocol with previous works in the low-dimensional balanced setting.

Implementation. Our end-to-end implementation[2] is written in C++ and utilizes the libOTe library [33] for performing OT Extension, the volePSI library [30] for instantiating the required OKVS scheme, and the cryptoTools library [32] to implement symmetric cryptographic primitives, e.g., PRNG. To instantiate the required OPRF primitive, we implement the construction of [18], replacing the polynomial interpolation technique with the same (state-of-the-art) OKVS scheme provided by the volePSI library. For security, we selected $\kappa = 128$ as the security parameter and $\lambda = 40$ as the statistical parameter, in line with standard practices in the literature.

Environment. We ran our benchmarks over a single thread on our server with the following hardware configuration: AMD EPYC 74F3 (3.2 GHz base clock, up to 4.0 GHz) and 256 GB RAM. As

[2]https://github.com/asu-crypto/daOT-fuzzyPSI

Table 2: Performance Comparison of Fuzzy PSI Protocols: The set size $m = n$, the dimension d, and threshold δ. GC refers to garbled circuit. (ERROR) indicates an error occurred when executing the benchmark for the specific parameter setting. (-) indicates no implementation was provided for the specific parameter setting or, in the case of GC, the experiment took more than 1 hour. (*) indicates parameter settings for the L_2 metric we did not measure, given that our protocol does not support it.

	n	Prot.	Communication Cost (MBs)						Runtime (Seconds)					
			(d, δ)											
			(2, 10)	(6, 10)	(10, 10)	(2, 30)	(6, 30)	(10, 30)	(2, 10)	(6, 10)	(10, 10)	(2, 30)	(6, 30)	(10, 30)
L_∞	2^8	Naive GC	330.14	994.14	1658.14	330.14	994.14	1658.14	2.48	7.33	12.27	2.45	7.36	12.54
		[38]	2.77	132	3520	8.27	396	$> 10^4$	2.15	102.46	2831.27	6.27	299.29	8464.17
		[10]	7.52	22.1	**36.8**	21.4	63.9	106	2.20	6.24	10.32	5.36	15.9	26.56
		[31]	1.77	21.2	142	1.77	21.2	142	–	–	–	–	–	–
		Ours	**0.17**	**2.46**	54.50	**0.17**	**2.46**	**54.50**	**0.12**	**0.43**	**7.07**	**0.12**	**0.43**	**7.03**
	2^{12}	Naive GC	84512.01	254496.01	-	84512.01	254496.01	-	636.72	2666.47	-	711.63	2006.82	-
		[38]	44.3	2112	$> 10^4$	132	> 6000	$> 10^5$	34.77	1682.01	ERROR	100.63	5003.74	ERROR
		[10]	120	354	922	343	1022	1702	37.84	108.42	175.92	89.17	271.75	448.05
		[31]	28.3	340	2265	28.3	340	2265	–	–	–	–	–	–
		Ours	**2.21**	**38.46**	**869.63**	**2.21**	**38.46**	**869.63**	**1.76**	**9.29**	**129.17**	**1.78**	**9.37**	**125.02**
	2^{16}	Naive GC	-	-	-	-	-	-	-	-	-	-	-	-
		[38]	708	$> 10^4$	$> 10^5$	2116	$> 10^4$	$> 10^6$	563.89	ERROR	ERROR	1655.61	ERROR	ERROR
		[10]	1924	5665	**9408**	5488	16358	27228	635.124	1772.58	2915.02	1509.90	ERROR	ERROR
		[31]	453	5436	36239	453	5436	36239	–	–	–	–	–	–
		Ours	**34.24**	**612.87**	13906.80	**34.24**	**612.87**	**13906.80**	**31.75**	**171.14**	**2441.31**	**32.18**	**169.35**	**2482.36**
L_1	2^8	Naive GC	457.54	1377.25	2293.74	457.58	1377.34	2293.65	3.18	11.08	18.35	3.12	10.94	17.97
		[38]	2.85	132	3520	8.51	396	$> 10^4$	2.15	–	–	6.29	–	–
		[10]	7.5	21.8	**36.4**	21.3	63.2	105	2.44	6.48	10.69	5.80	16.63	26.97
		[31]	1.78	21.3	142	1.78	21.3	142	–	–	–	–	–	–
		Ours	**0.33**	**2.99**	55.27	**0.54**	**3.63**	**56.43**	**0.12**	**0.43**	**7.05**	**0.13**	**0.44**	**7.11**
	2^{12}	Naive GC	117135.53	352566.74	-	117144.29	352557.29	-	1207.55	3081.34	-	1381.35	3225.4	-
		[38]	45.6	2113	$> 10^4$	132	> 6000	$> 10^5$	34.97	–	–	101.64	–	–
		[10]	120	351	**589**	340	1024	1703	40.87	111.91	181.98	94.93	281.14	464.05
		[31]	28.4	341	2274	28.4	341	2274	–	–	–	–	–	–
		Ours	**4.67**	**46.74**	881.89	**8.05**	**56.86**	**900.27**	**1.83**	**9.36**	**147.94**	**1.9**	**9.73**	**126.94**
	2^{16}	Naive GC	-	-	-	-	-	-	-	-	-	-	-	-
		[38]	730	$> 10^4$	$> 10^5$	2179	$> 10^4$	$> 10^6$	564.26	–	–	1659.55	–	–
		[10]	1919	5685	**9427**	5513	16382	27253	706.08	1902.36	3055.09	1660.61	ERROR	ERROR
		[31]	455	5457	36390	455	5457	36390	–	–	–	–	–	–
		Ours	**73.44**	**745.08**	14102.6	**127.30**	**906.58**	**14396.2**	**32.84**	**173.19**	**2393.49**	**35.08**	**180.27**	**2481.96**
L_2	2^8	Naive GC	17281.73	*	*	17281.69	*	*	54.45	*	*	54.36	*	*
		[38]	3.55	*	*	15.3	*	*	2.14	*	*	6.23	*	*
		[10]	7.59	*	*	21.4	*	*	2.63	*	*	5.82	*	*
		[31]	4.63	*	*	4.63	*	*	–	*	*	–	*	*
		Ours	**0.33**	*	*	**0.54**	*	*	**0.12**	*	*	**0.12**	*	*
	2^{12}	Naive GC	-	*	*	-	*	*	-	*	*	-	*	*
		[38]	56.9	*	*	245	*	*	34.47	*	*	99.86	*	*
		[10]	122	*	*	347	*	*	43.76	*	*	99.17	*	*
		[31]	73.8	*	*	73.8	*	*	–	*	*	–	*	*
		Ours	**4.67**	*	*	**8.05**	*	*	**1.84**	*	*	**1.82**	*	*
	2^{16}	Naive GC	-	*	*	-	*	*	-	*	*	-	*	*
		[38]	911	*	*	3919	*	*	560.01	*	*	1649.77	*	*
		[10]	1964	*	*	5549	*	*	748.30	*	*	1735.24	*	*
		[31]	1191	*	*	1181	*	*	–	*	*	–	*	*
		Ours	**73.44**	*	*	**127.30**	*	*	**33.2**	*	*	**35.3**	*	*

in [37], we did not account for network latency when measuring the runtime of our implementations. This should not be an issue, as measurements show that our protocol has a significant advantage in communication cost compared to the others.

Concrete Performance and Comparison. We evaluate fuzzy-PSI protocols for all three distance metrics (i.e., L_∞, L_1, L_2) and report their performance in Table 2. For a fair comparison with the most recent work [31], we adopt their parameters. Additionally, to facilitate comparisons with [9, 31, 37], we assume that Alice's points are separated by at least 4δ. In all comparisons (similar to [31]), we assume that both parties have sets of equal size, evaluating performance across set sizes $\{2^8, 2^{12}, 2^{16}\}$. Beyond input set sizes, we examine these constructions under varying dimensions d and distance threshold δ. Specifically, for both L_∞ and L_1 based fuzzy PSI, we evaluate the protocols with $d \in \{2, 6, 10\}$ and $\delta \in \{10, 30\}$, while for metric L_2, our comparisons focus on dimension $d = 2$ and $\delta \in \{10, 30\}$. To enable a comprehensive comparison, we also included the performance of naive garbled-circuit-based solution, implemented using the EMP-sh2pc library [1]. In this solution, we use standard garbled circuits to perform pair-wise distance comparison between all points of the sender and the receiver to compute the output, leading to quadratic complexity.

We also used the reported numbers from [31] to compare concrete communication costs for all other protocols. For computational costs, we evaluated open-source implementations from

[9, 37]. We did not include [31] in the computational cost comparison, as the authors provided no concrete implementation. Our solution significantly outperforms prior work, achieving better runtime performance for all the parameters included in our comparison and, in some cases, achieving a 54× improvement. Regarding communication costs, our solution improves on previous works for all settings except for the case where we have $d = 10, \delta = 10$. In this case, however, our solution requires less 2× communication than the best solution. For the settings where our solution is the better option communication-wise, we can sometimes achieve up to an 14× improvement.

ACKNOWLEDGMENTS.

Lucas Piske and Ni Trieu were partially supported by NSF award #2115075, and ARPA-H SP4701-23-C-0074. Vladmir Kolesnikov was partially supported by Visa research award and NSF awards CNS-2246354, and CCF-2217070. Vassilis Zikas was supported in part by NSF Grant No. 2448339, by Sunday Group, Inc., and by the United States-Israel Binational Science Foundation (BSF) through Grant No. 2020277. Jaspal Singh was supported in part by Sunday Group, Inc., and the BSF Grant No. 2020277.

REFERENCES

[1] GitHub - emp-toolkit/emp-sh2pc: Semi-honest Two Party Computation Based on Garbled Circuits. — github.com. https://github.com/emp-toolkit/emp-sh2pc.git. [Accessed 16-04-2025].
[2] Ian F. Blake and Vladimir Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 515–529. Springer, Berlin, Heidelberg, December 2004.
[3] Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. All-or-nothing disclosure of secrets. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 234–238. Springer, 1986.
[4] Anrin Chakraborti, Giulia Fanti, and Michael K Reiter. {Distance-Aware} private set intersection. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 319–336, 2023.
[5] Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-PSI with linear complexity via relaxed batch OPPRF. Cryptology ePrint Archive, Paper 2021/034, 2021.
[6] 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.
[7] Jiahui Gao, Truong Son Nguyen, and Ni Trieu. Toward a practical multi-party private set union. *Proceedings on Privacy Enhancing Technologies*, 2024(4):622–635, 2024.
[8] Jiahui Gao, Ni Trieu, and Avishay Yanai. Multiparty private set intersection cardinality and its applications. In *Proceedings on Privacy Enhancing Technologies (PETS)*, 2024.
[9] 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.
[10] Ying Gao, Lin Qi, Xiang Liu, Yuanchao Luo, and Longxin Wang. Efficient fuzzy private set intersection from fuzzy mapping. *Cryptology ePrint Archive*, 2024.
[11] Gayathri Garimella, Benjamin Goff, and Peihan Miao. Computation efficient structure-aware PSI from incremental function secret sharing. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 309–345. Springer, Cham, August 2024.
[12] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private set operations from oblivious switching. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 591–617. Springer, Cham, May 2021.
[13] 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.
[14] 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.
[15] 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.
[16] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 370–389. IEEE, 2020.
[17] 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, August 2013.
[18] 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.
[19] Vladimir Kolesnikov, Jesper Buus Nielsen, Mike Rosulek, Ni Trieu, and Roberto Trifiletti. DUPLO: Unifying cut-and-choose for garbled circuits. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 3–20. ACM Press, October / November 2017.
[20] Tancrède Lepoint, Sarvar Patel, Karn Seth, Mariana Raykova, and Ni Trieu. Private join and compute from pir with default. In *Advances in Cryptology – ASIACRYPT 2021*, volume 13090 of *Lecture Notes in Computer Science*, pages 605–635. Springer, 2021.
[21] Yehuda Lindell. *How to Simulate It – A Tutorial on the Simulation Proof Technique*, pages 277–346. Springer International Publishing, Cham, 2017.
[22] Rasoul Akhavan Mahdavi, Nils Lukas, Faezeh Ebrahimianghazani, Thomas Humphries, Bailey Kacsmar, John Premkumar, Xinda Li, Simon Oya, Ehsan Amjadian, and Florian Kerschbaum. PEPSI: Practically efficient private set intersection in the unbalanced setting. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 6453–6470, Philadelphia, PA, August 2024. USENIX Association.
[23] Ian McQuoid, Mike Rosulek, and Lawrence Roy. Batching base oblivious transfers. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 281–310. Springer, Cham, December 2021.
[24] Dimitris Mouris, Daniel Masny, Ni Trieu, Shubho Sengupta, Prasad Buddhavarapu, and Benjamin Case. Delegated private matching for compute. In *Proceedings on Privacy Enhancing Technologies (PETS)*, 2024.
[25] Ofri Nevo, Ni Trieu, and Avishay Yanai. Simple, fast malicious multiparty private set intersection. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1151–1165. ACM Press, November 2021.
[26] 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.
[27] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 739–767. Springer, Cham, May 2020.
[28] Lucas Piske, Jeroen Graaf, Anderson CA Nascimento, and Ni Trieu. Shared ot and its applications to unconditional secure integer equality, comparison and bit-decomposition. *Cryptology ePrint Archive*, 2024.
[29] Michael Rabin. How to exchange secrets by oblivious transfer, 1981.
[30] Visa Research. volepsi: A protocol for private set intersection. https://github.com/Visa-Research/volepsi/blob/main/volePSI/, 2023. Accessed: 2025-01-07.
[31] David Richardson, Mike Rosulek, and Jiayu Xu. Fuzzy psi via oblivious protocol routing. *Cryptology ePrint Archive*, 2024.
[32] Peter Rindal et al. cryptotools: A cryptographic library for secure computation. https://github.com/ladnir/cryptoTools, 2023. Accessed: 2025-01-07.
[33] Peter Rindal et al. libote: A fast and portable oblivious transfer library. https://github.com/osu-crypto/libOTe, 2023. Accessed: 2025-01-07.
[34] Mike Rosulek and Ni Trieu. Compact and malicious private set intersection for small sets. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1166–1181. ACM Press, November 2021.
[35] Phillipp Schoppmann, Adrià Gascón, Mariana Raykova, and Benny Pinkas. Make some ROOM for the zeros: Data sparsity in secure distributed machine learning. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1335–1350. ACM Press, November 2019.
[36] 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 *30th USENIX Security Symposium (USENIX Security 21)*, pages 911–928. USENIX Association, August 2021.
[37] Aron van Baarsen and Sihang Pu. Fuzzy private set intersection with large hyperballs. Eurocrypt, 2024.
[38] Aron van Baarsen and Sihang Pu. Fuzzy private set intersection with large hyperballs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 340–369. Springer, 2024.

A CONCLUSION

This work introduces a new modular framework for fuzzy PSI, primarily constructed using efficient symmetric-key primitives. At its core is a novel OT variant called Distance-aware Random OT (daROT). Compared to prior approaches based on FSS or asymmetric cryptographic techniques, our protocol achieves a 14× improvement in communication cost and a 54× in computation cost for input sets of size from 2^8 to 2^{16}. Furthermore, it operates under weaker cryptographic assumptions than existing solutions.

While our presented solution is in the unique grid cell assumption, they can be extended to support arbitrary sender inputs as well. In the assumption-free setting, if the sender holds at most $t > 1$ points per grid cell, it can decompose its input set S into t disjoint sets S_i (for $i \in [t]$) such that each point of S is randomly mapped to one set S_i conditioned on each S_i containing at most one point per grid cell. Further the sender and receiver can run t instances of our fuzzy PSI protocol based on unique grid cell assumption, where receiver inputs its original set and the sender inputs set S_i (padded with dummy elements to make set size $|S|$) in the i^{th} fuzzy PSI call. The security of this scheme directly follows from our proposed fuzzy PSI construction and the leakage of the additional parameter t. We leave it as future work to optimize our construction for the assumption-free case where the multiplicative t overhead can be avoided.

B PRELIMINARIES: DETAILS

B.1 Secure Semi-Honest Model

We use the standard notion of security in the presence of semi-honest adversaries. Let π be a protocol for computing the a (probabilistic) polynomial time functionality $f(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))$, where party P_i has input x_i and it receives as output $f_1(x_1, x_2)$. For party P_i, let $\mathsf{view}_i(1^\kappa, x_1, x_2)$ denote the view of party P_i during an honest execution of π on inputs x_1, x_2 and security parameter κ, and $\mathsf{out}_i(1^\kappa, x_1, x_2)$ denote the output received by P_i from protocol π. We also use $\mathsf{out}(1^\kappa, x_1, x_2) = (\mathsf{out}_1(1^\kappa, x_1, x_2), \mathsf{out}_2(1^\kappa, x_1, x_2))$ as shorthand to denote the protocol's joint output of both parties.

We have the formal definition of semi-honest security (see Definition B.1) as defined by [21]. Given the nature of the functionalities presented in this work, we use the more general definition, which supports non-deterministic functionalities.

Definition B.1. [21] Let κ denote a computational security parameter and $\cong_\kappa$ denote computational indistinguishability. A 2-party protocol π securely realizes a probabilistic polynomial time functionality f against static semi-honest adversaries if there exists a probabilistic polynomial-time simulator Sim such that, for all input pairs x_1, x_2 and all $i \in \{1, 2\}$:

$$(\mathsf{Sim}(1^\kappa, i, x_i, f_i(x_1, x_2)), f(x_1, x_2)) \cong_\kappa (\mathsf{view}_i(1^\kappa, x_1, x_2), \mathsf{out}(1^\kappa, x_1, x_2)).$$

Theorem 2. *For domain $\mathcal{U}^d$, grid parameter δ, any $X, Y \subset \mathcal{U}^d$ where $\rho_c^X = 1$. Let $D_X \leftarrow \mathsf{cellhash}(X, \rho_\eta^Y)$ and $D_Y \leftarrow \mathsf{sphash}(Y)$. Then the following holds for any $x \in X, y \in Y$ and $p \in [0, \infty]$*

- *If $\mathtt{dist}_p(x, y) \le \delta$: there exist $c \in C$ such that $(c, \mathsf{trunc}(x)) \in D_X$ and $(c, \mathsf{trunc}(y)) \in D_Y$*
- *If $\mathtt{dist}_p(x, y) > \delta$: either there exist no c such that $(c, \mathsf{trunc}(x)) \in D_X$ and $(c, \mathsf{trunc}(y)) \in D_Y$, or $\mathtt{dist}_p(\mathsf{trunc}(x), \mathsf{trunc}(y)) > \delta$*

B.2 Oblivious Key-Value Store (OKVS)

Definition B.2. [13] An OKVS is parameterized by a set $\mathcal{K}$ of keys, a set $\mathcal{V}$ of values, and a set of functions H, and consists of two algorithms:

- $\mathsf{OKVS.Encode}_H$ takes as input a set of (k_i, v_i) key-value pairs and outputs an object D (or, with statistically small probability, an error indicator $\perp$).
- $\mathsf{OKVS.Decode}_H$ takes as input an object D, a key k, and outputs a value v.

An OKVS is **correct** if, for all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys:

$$\big((k, v) \in A\big) \wedge \big(\{\perp, D\} \leftarrow \mathsf{OKVS.Encode}_H(A)\big) \implies \mathsf{OKVS.Decode}_H(D, k) = v$$

An OKVS is **oblivious** if, for all distinct $\{k_1^0, \ldots, k_n^0\}$ and all distinct $\{k_1^1, \ldots, k_n^1\}$, if OKVS.Encode does not output $\perp$ for $(k_1^0, \ldots, k_n^0)$ or $(k_1^1, \ldots, k_n^1)$, then the output of $\mathcal{R}(k_1^0, \ldots, k_n^0)$ is computationally indistinguishable to that of $\mathcal{R}(k_1^1, \ldots, k_n^1)$, where:

$\underline{\mathcal{R}(k_1, \ldots, k_n)}$:
  for $i \in [n]$: do $v_i \leftarrow \mathcal{V}$
  return $\mathsf{OKVS.Encode}(\{(k_1, v_1), \ldots, (k_n, v_n)\})$

In our proofs, we assume a more general obliviousness property: the OKVS output reveals only the identities of keys whose corresponding values are not randomly chosen. For a formal definition of this generalized obliviousness property, we refer the reader to [31].

In this paper, we will omit the underlying parameter H whenever the context is clear and unambiguous.

B.3 Oblivious Pseudorandom Function (OPRF)

Definition B.3. A Pseudorandom Function (PRF) consists of the following two PPT algorithms for a domain $\mathcal{D}$ and a range $\mathcal{R}$:

- $\mathsf{KeyGen}(1^\kappa) \to \mathsf{k}$: Given a security parameter κ, this algorithm generates a PRF key k. We often omit the security parameter argument when it is clear from context.
- $F_\mathsf{k}(x) \to y$: Evaluates the PRF on input $x \in \mathcal{D}$ using the key k, giving output $y \in \mathcal{R}$.

A PRF is secure if, for all, PPT distinguishes $\mathcal{A}$, there is a negligible function negl such that:

$$\left|\Pr[\mathcal{A}^{F_\mathsf{k}(\cdot)}(1^\kappa)] - \Pr[\mathcal{A}^{f(\cdot)}(1^\kappa)]\right| \le \mathsf{negl}(\kappa),$$

where both probabilities are taken over the randomness of $\mathsf{k} \leftarrow \mathsf{KeyGen}(1^\kappa)$ and $\mathcal{A}$, and the second one is also taken over the uniform choice of f from the family of functions $f : \mathcal{D} \to \mathcal{R}$.

Ideal Functionality $\mathcal{F}_{\mathsf{OPRF}}$

Parameters: Two parties: sender S and receiver R, a PRF scheme F for domain $\mathcal{D}$ and range $\mathcal{R}$, and bound $t \in \mathbb{N}$.

Behavior:

- Wait for input ordered set $X = \{x_0, \ldots, x_{t-1}\} \subseteq \mathcal{D}$ of size t from R.
- Sample k using KeyGen and give it to S.
- Give $(F_k(x_0), \ldots, F_k(x_{t-1}))$ to R.

C ASYMPTOTIC EFFICIENCY OF OUR PROTOCOLS

C.1 Our da-ROT Protocol

We assume computing the message vectors defined by Equations 1 and 2 of Figure 2 take $O(v)$ time and the ones defined by Equation 3 takes $O(M)$ time. To instantiate the $\mathcal{F}_{\text{SOT}}$ functionality we use the protocol proposed in [28], which, when amortized, takes $O(\log_2(N) + N \cdot \log(M))$ time to execute and requires $O(\log N + N \cdot \log M)$ bits of communication, where N is the message vector size and the messages are elements mod-M.

Based on these assumptions, we can conclude that our protocol $\Pi_{\text{da-ROT}}$ takes $O(d \cdot (\log v + v \cdot \log M) + \log M + M \cdot \log u)$ time to execute and requires $O(d \cdot (\log v + v \cdot \log M) + \log M + M \cdot \log u)$ bits of communication, where $M = d + 1$ if $\mathsf{p} = \infty$, and $M = d \cdot (\delta + 1) + 1$ for $\mathsf{p} \in \{1, 2\}$.

C.2 Our SpSOT Protocol

We assume mod-M addition and subtractions can be performed in constant time, cshift_N is done in $O(N)$ time, and that OKVS encoding and batch decoding take $O(n \cdot \lambda)$ time, with the encoded OKVS structure having $O(n * \ell)$ bits in size, for n keys and elements of bit-size ℓ.

To instantiate $\mathcal{F}_{\text{OPRF}}$, we use the protocol proposed in [26], replacing the polynomial with an OVKVS scheme. Assuming the PRF used by this OPRF protocol executes in $O(\kappa)$ time, the resulting OPRF protocol takes $O(n \cdot (\kappa^2 + \lambda))$ time to perform the oblivious evaluation of n points, $O(n \cdot (\kappa^2 + \lambda))$ to perform a batch of n non-oblivious evaluations of the PRF, and requires $O(n \cdot \kappa)$ bits of communication when amortized.

The protocol Π_{SpSOT} is composed of the following operations:

- 1 $\mathcal{F}_{\text{OPRF}}$ execution for PRF F and n^R oblivious evaluations, where the generated key is θ.
- $n^S \cdot N$ F_θ non-oblivious evaluations.
- 1 $\mathcal{F}_{\text{OPRF}}$ execution for PRF H and n^S oblivious evaluations, where the generated key is ϕ.
- n^R F_ϕ non-oblivious evaluations.
- n^S calls to cshift_N.
- 1 call to OKVS.Encode with $n^S \cdot N$ key-values pairs.
- n^R calls to OKVS.Decode.
- $O(n^S \cdot N + n^R)$ mod-M addition/subtraction operations.

Based on our previously stated assumptions, the protocol Π_{SpSOT} then requires $O((n^R + n^S \cdot N) \cdot (\kappa^2 + \lambda))$ time to execute, and $O(n^S \cdot (N \cdot \log M + \kappa) + n^R \cdot \kappa))$ bits of communication.

C.3 Our Sp-daROT Protocol

We now describe the asymptotic analysis of the resulting Sparse da-ROT protocol when our compiler is provided with the da-ROT protocol defined in Section 3.1 as input and $\mathcal{F}_{\text{SpSOT}}$ is instantiated by Π_{SpSOT}.

Independent of metric p, our da-ROT protocol is composed of d 1-oo-v SOTs with messages in $\mathbb{Z}_M$, a single 1-oo-M SOT with messages in $\mathbb{Z}_u$ and local computations. The local computations don't impact the final asymptotic efficiency, so we will ignore them in this analysis. Given the previously described asymptotic efficiency of Π_{SpSOT}, our compiler gives us a protocol with computational efficiency of $O(d \cdot (n_1 + n_0 \cdot v) \cdot (\kappa^2 + \lambda) + (n_1 + n_0 \cdot M) \cdot (\kappa^2 + \lambda))$ and communication efficiency of $O(d \cdot n_0 \cdot (v \cdot \log M + \kappa) + n_0 \cdot (M \cdot \log u + \kappa) + (d + 1) \cdot n_1 \cdot \kappa))$, where M depends on p as described in Figure 2.

C.4 Our Fuzzy PSI Framework

All three protocols described in Figure 5 share the costs incurred by executing the first three steps. Based on the asymptotic analysis of $\Pi_{\text{Sp-daROT}}$ presented in Section 4.1, given set sizes $|A'| = n \cdot 2^d$ and $|B'| = m$, we know the first three steps will have a computational cost of $O(d \cdot (n \cdot 2^d + m \cdot 2^{t^*}) \cdot (\kappa^2 + \lambda) + (n \cdot 2^d + m \cdot M) \cdot (\kappa^2 + \lambda))$ and communication cost of $O(d \cdot m \cdot (2^{t^*} \cdot \log M + \kappa) + m \cdot (M \cdot \kappa) + (d + 1) \cdot n \cdot 2^d \cdot \kappa))$, where M depends on metric p as described in Figure 2 and $t^* = \lceil \log(6\delta) \rceil$.

Fuzzy PSI. Aside from the cost of the first three steps, the fuzzy PSI protocol requires Bob to encrypt $|B_{\text{out}}| = m \cdot \rho_\eta^A$ values and to encode an OKVS structure D with $|B_{\text{out}}| = m \cdot \rho_\eta^A$ items and requires Alice to query OKVS D $|A_{\text{out}}| = n \cdot \rho_\eta^A \cdot 2^d$ times and to decrypt $|A_{\text{out}}| = n \cdot \rho_\eta^A \cdot 2^d$ ciphertexts. Assuming the same OKVS asymptotic efficiency as in Section 3.1 and assuming that encryption/decryption takes $O(\kappa)$ time, both the total computational and communication asymptotic costs are dominated by the first three steps. This leaves us with a fuzzy PSI protocol that has a total computational cost of $O(d \cdot \rho_\eta^A \cdot (n \cdot 2^d + m \cdot 2^{t^*}) \cdot (\kappa^2 + \lambda) + \rho_\eta^A \cdot (n \cdot 2^d + m \cdot M) \cdot (\kappa^2 + \lambda))$ and a total communication cost of $O(d \cdot \rho_\eta^A \cdot m \cdot (2^{t^*} \cdot \log M + \kappa) + m \cdot (M \cdot \kappa) + (d + 1) \cdot n \cdot \rho_\eta^A \cdot 2^d \cdot \kappa))$, where $2^{t^*} \in O(\delta)$.

Fuzzy Cardinality. Aside from the first three protocol steps, the two parties build the sets X and Y and then execute the $\mathcal{F}_{\text{PSICard}}$ using these two sets as input. This gives a total computational cost of $O(d \cdot \rho_\eta^A \cdot (n \cdot 2^d + m \cdot 2^{t^*}) \cdot (\kappa^2 + \lambda) + (n \cdot \rho_\eta^A \cdot 2^d + m \cdot M) \cdot (\kappa^2 + \lambda) + \text{cmp-cost}_{\text{card}})$ and a communication cost of $O(d \cdot m \cdot \rho_\eta^A \cdot (2^{t^*} \cdot \log M + \kappa) + m \cdot (M \cdot \kappa) + (d + 1) \cdot n \cdot \rho_\eta^A \cdot 2^d \cdot \kappa) + \text{cmm-cost}_{\text{card}})$, where $\text{cmp-cost}_{\text{card}}$ and $\text{cmm-cost}_{\text{card}}$ are the computational and communication costs of the protocol used to instantiate $\mathcal{F}_{\text{fuzzyCard}}$, respectively.

Fuzzy Join. In the three protocol steps exclusive to the fuzzy join protocol, the two parties build two dictionaries and execute the $\mathcal{F}_{\text{fuzzyJoin}}$ functionality, providing these dictionaries as input. This leaves us with a fuzzy join protocol with a total computational cost of $O(d \cdot \rho_\eta^A \cdot (n \cdot 2^d + m \cdot 2^{t^*}) \cdot (\kappa^2 + \lambda) + \rho_\eta^A \cdot (n \cdot 2^d + m \cdot M) \cdot (\kappa^2 + \lambda) + \text{cmp-cost}_{\text{join}})$ and a total communication cost of $O(d \cdot \rho_\eta^A \cdot m \cdot$

$(2^{t^*} \cdot \log M + \kappa) + m \cdot (M \cdot \kappa) + (d+1) \cdot n \cdot \rho_\eta^A \cdot 2^d \cdot \kappa) + \text{cmm-cost}_{\text{join}})$, where $\text{cmp-cost}_{\text{join}}$ and $\text{cmm-cost}_{\text{join}}$ are the computational and communication costs of the protocol used to instantiate $\mathcal{F}_{\text{fuzzyJoin}}$, respectively.

D SECURITY PROOF

D.1 Security for Sparse SOT

Theorem 3. *Protocol Π_{SpSOT} securely realizes the Sparse SOT functionality $\mathcal{F}_{\text{SpSOT}}$ against a PPT semi-honest adversary in the $\mathcal{F}_{OPRF}$-hybrid model.*

Proof. Denote the inputs provided by R and S to the functionality and the protocol as $x_R = (I^R, (\llbracket c^{(i)} \rrbracket_M^R)_{i \in I^R})$ and $x_S = (I^S, (\boldsymbol{m}^{(i)}, \llbracket c^{(i)} \rrbracket_M^S)_{i \in I^S})$, respectively. Let $\boldsymbol{y}^R \in \mathbb{Z}_M^{n^R}$ and $\boldsymbol{y}^S \in \mathbb{Z}_M^{n^S}$ represent the output vectors returned by the protocol $\mathcal{F}_{\text{SpSOT}}$ to R and S, respectively. Similarly, let $\boldsymbol{g}^R \in \mathbb{Z}_M^{n^R}$ and $\boldsymbol{g}^S \in \mathbb{Z}_M^{n^S}$ denote the output vectors returned by the functionality Π_{SpSOT} to R and S, respectively.

Corrupt Receiver. From the protocol description, the R's protocol transcript $\text{view}_R(1^\kappa, x_R, x_S)$ consists of the following elements in order: the output vector $\boldsymbol{f} \in \mathbb{Z}_M^{n^R}$ received from $\mathcal{F}_{\text{OPRF}}$ in step 1, the key ϕ received from $\mathcal{F}_{\text{OPRF}}$ at step 3, and the OKVS D received from S at step 8.

$$\text{view}_R(1^\kappa, x_R, x_S) = (\boldsymbol{f}, \phi, D)$$

The simulator Sim starts by computing $D' \leftarrow \text{OKVS.Encode}(E')$ where $E' = \{((i,0), v^{(i)}) \mid i \in I'^S\}$ (here, we include a dummy value 0 as the second component of the tuple in the keyword for the OKVS.), with $I'^S \subseteq \mathcal{I}$ being a uniformly sampled set of size n^S and $v^{(i)} \in_R \mathbb{Z}_M$ for every $i \in I'^S$. Next, Sim samples a key ϕ' to the PRF H. To simulate the output $\boldsymbol{f}$ that R receives at step 1 from $\mathcal{F}_{\text{OPRF}}$, Sim waits for the input query-point set and answers with vector $\boldsymbol{f}' \in \mathbb{Z}_M^{n^R}$, which is computed as follows:

$$f'_j = y_j^R - \text{OKVS.Decode}(D', (i_j^R, \llbracket c^{(i_j^R)} \rrbracket_N^R)) - H_\phi(i_j^R)$$

To simulate the key ϕ that R receives as output from $\mathcal{F}_{\text{OPRF}}$ in step 3 and the OKVS R obtains from S in step 8, Sim sends ϕ' and D' at the corresponding steps. Next, we prove that Sim satisfies the following equation for all possible inputs of the parties.

$$\text{Sim}(1^\kappa, R, x_R, \boldsymbol{y}^R) \cong_\kappa \text{view}_R(1^\kappa, x_R, x_S)$$

From the definition of $\mathcal{F}_{\text{SpSOT}}$, we have that $\boldsymbol{y}^R \in_R \mathbb{Z}_M^{n^R}$, which directly implies that $\boldsymbol{f}' \in_R \mathbb{Z}^{n^R} M$, and consequently $\boldsymbol{f}' \cong_\kappa \boldsymbol{f}$. Similarly, from the definition of $\mathcal{F}$OPRF, it follows that $\phi' \cong_\kappa \phi$. Next, we prove that $D \cong_\kappa D'$, beginning with an analysis of the pseudorandomness of D with respect to R. Considering two cases.

(1) $\mathbf{i \in I^S \setminus I^R}$: Since the value of $\boldsymbol{w}^{(i)} \in \mathbb{Z}_M^{n^S}$ is given by

$$w_j^{(i)} \leftarrow u_j^{(i)} - \llbracket z^{(i)} \rrbracket_M^S - F_\theta(i,j) - h^{(i)}$$

and R only knows evaluations $\boldsymbol{f} = (F_\theta(i, \llbracket c^{(i)} \rrbracket_N^R))_{i \in I^R}$ of F_θ, it is clear that that $\boldsymbol{w}^{(i)}$ is pseudorandom to R.

(2) $\mathbf{i \in I^S \cap I^R}$: Given how $\boldsymbol{w}^{(i)}$ is computed and the points queried during step 1 of Π_{SpSOT}, the $w_j^{(i)}$ appears pseudorandom to R for every $j \neq \llbracket c^{(i)} \rrbracket_N^R$. Additionally, $w^{(i)}_{\llbracket c^{(i)} \rrbracket_N^R}$ is also pseudorandom to R because S samples $\llbracket z^{(i)} \rrbracket_M^S$ uniformly from $\mathbb{Z}_M$.

Since $D \leftarrow \text{OKVS.Encode}(E)$ where $E = \{((i,j), w_j^{(i)}) \mid (i,j) \in I^S \times \mathbb{Z}_N\}$, and given that $\boldsymbol{w}^{(i)}$ is pseudorandom to R for every $i \in I^S$, combined with the oblivious property of OKVS schemes, we can conclude that $D' \cong_\kappa D$ with respect to R.

Next, we show that the following holds for every possible input.

$$(\text{Sim}(1^\kappa, R, x_R, \boldsymbol{y}^R), \boldsymbol{y}^R) \cong_\kappa (\text{view}_R(1^\kappa, x_R, x_S), \boldsymbol{g}^R) \tag{4}$$

From the description of Π_{SpSOT}, we can see that $\boldsymbol{g}^R$ is defined by

$g_j^R = \llbracket z^{(i)} \rrbracket_M^R = \text{OKVS.Decode}(D, (i, \llbracket c^{(i)} \rrbracket_N^R)) + f^{(i)} + H_\phi(i)$, where $i = i_j^R$.

By replacing $D, \boldsymbol{f}, \phi$ for the their simuled counterparts $D', \boldsymbol{f}', \phi'$, we have

$\text{OKVS.Decode}(D', (i, \llbracket c^{(i)} \rrbracket_N^R)) + f'^{(i)} + H_{\phi'}(i) = y_j^R$, where $i = i_j^R$.

This completes the proof of Equation (4). Finally, we prove:

$$(\text{Sim}(1^\kappa, R, x_R, \boldsymbol{y}^R), (\boldsymbol{y}^R, \boldsymbol{y}^S)) \cong_\kappa (\text{view}_R(1^\kappa, x_R, x_S), (\boldsymbol{g}^R, \boldsymbol{g}^S)) \tag{5}$$

From the definition of $\mathcal{F}_{\text{SpSOT}}$, we have that $\boldsymbol{y}^S \in_R \mathbb{Z}_M^{n^S}$ and

$y_j^R + y_k^S = m^{(i)}_{c^{(i)}}$, for every $i_j^R \in I^R$ and $i_k^S \in I^S$, such that $i = i_j^R = i_k^S$.

We now proceed to prove the same holds for $\boldsymbol{g}^R$ and $\boldsymbol{g}^S$. Let $i \in I^R \cap I^S$. From the protocol description, we have

$$g_j^R = \llbracket z^{(i_j^R)} \rrbracket_M^R, \forall i_j^R \in I^R \quad \text{and} \quad g_k^S = \llbracket z^{(i_k^S)} \rrbracket_M^S, \forall i_k^S \in I^S$$

Additionally,

$$\begin{aligned}
&\llbracket z^{(i)} \rrbracket_M^R \\
&= q^{(i)}_{\llbracket c^{(i)} \rrbracket_N^R} + f^{(i)} + H_\phi(i) \\
&= q^{(i)}_{\llbracket c^{(i)} \rrbracket_N^R} + F_\theta(i, \llbracket c^{(i)} \rrbracket_N^R) + H_\phi(i) \\
&= \text{OKVS.Decode}(i, \llbracket c^{(i)} \rrbracket_N^R) + F_\theta(i, \llbracket c^{(i)} \rrbracket_N^R) + H_\phi(i) \\
&= w^{(i)}_{\llbracket c^{(i)} \rrbracket_N^R} + F_\theta(i, \llbracket c^{(i)} \rrbracket_N^R) + H_\phi(i) \\
&= u^{(i)}_{\llbracket c^{(i)} \rrbracket_N^R} - \llbracket z^{(i)} \rrbracket_M^S - F_\theta(i, \llbracket c^{(i)} \rrbracket_N^R) - H_\phi(i) + F_\theta(i, \llbracket c^{(i)} \rrbracket_N^R) + H_\phi(i) \\
&= u^{(i)}_{\llbracket c^{(i)} \rrbracket_N^R} - \llbracket z^{(i)} \rrbracket_M^S \\
&= m^{(i)}_{\llbracket c^{(i)} \rrbracket_N^R + \llbracket c^{(i)} \rrbracket_N^R \pmod N} - \llbracket z^{(i)} \rrbracket_M^S \\
&= m^{(i)}_{c^{(i)}} - \llbracket z^{(i)} \rrbracket_M^S
\end{aligned}$$

This implies that $g^S \in_R \mathbb{Z}_M^{n^S}$ and

$g_j^R + g_k^S = m^{(i)}_{c^{(i)}}$, for every $i_j^R \in I^R$ and $i_k^S \in I^S$, such that $i = i_j^R = i_k^S$.

This concludes our simulation proof for a corrupt receiver.

Corrupt Sender. The S's protocol transcript $\mathsf{view}_S(1^\kappa, x_R, x_S)$ consists of the PRF key θ that S receives at step 1, followed by the output vector $\boldsymbol{h} \in \mathbb{Z}_M^{n^S}$ that S receives in step 3.

$$\mathsf{view}_S(1^\kappa, x_R, x_S) = (\theta, \boldsymbol{h})$$

Sim simulates the output θ at step 1 by sampling and returning a key θ' to PRF F, and simulates the output vector $\boldsymbol{h}$ at step 3 by sampling $\boldsymbol{h}' \in_R \mathbb{Z}_M^{n^S}$ and returning it. From the definition of $\mathcal{F}_{\mathsf{OPRF}}$, it is straightforward to see the following holds for any inputs provided by the parties.

$$\mathsf{Sim}(1^\kappa, S, x_S, \boldsymbol{y}^S) \cong_\kappa \mathsf{view}_S(1^\kappa, x_R, x_S)$$

From the protocol description, the party S's output $\boldsymbol{g}^S$ is uniformly sampled from $\mathbb{Z}_M^{n^S}$. Thus, the next equation follows in a straightforward manner for any inputs provided by the parties.

$$(\mathsf{Sim}(1^\kappa, S, x_S, \boldsymbol{y}^S), \boldsymbol{y}^S) \cong_\kappa (\mathsf{view}_S(1^\kappa, x_R, x_S), \boldsymbol{g}^S)$$

Next, we complete our proof by showing the following for any inputs provided by the parties:

$$(\mathsf{Sim}(1^\kappa, S, x_S, \boldsymbol{y}^S), (\boldsymbol{y}^R, \boldsymbol{y}^S)) \cong_\kappa (\mathsf{view}_S(1^\kappa, x_R, x_S), (\boldsymbol{g}^R, \boldsymbol{g}^S)) \quad (6)$$

From the definition of $\mathcal{F}_{\mathsf{SpSOT}}$, we have that $\boldsymbol{y}^R \in_R \mathbb{Z}_M^{n^R}$ and

$y_j^R + y_k^S = m_{c^{(i)}}^{(i)}$, for every $i_j^R \in I^R$ and $i_k^S \in I^S$, such that $i = i_j^R = i_k^S$.

Similar to the case of a corrupt receiver, we have that $\boldsymbol{g}^R \in \mathbb{Z}_M^{n^R}$ and

$g_j^R + g_k^S = m_{c^{(i)}}^{(i)}$, for every $i_j^R \in I^R$ and $i_k^S \in I^S$, such that $i = i_j^R = i_k^S$.

Moreover, g_j^R is pseudorandom to S for every $i_j^R \in I^R \setminus I^S$. Let $i \in I^R \setminus I^S$. As previously noted, we have that

$g_j^R = [\![z^{(i)}]\!]_M^R = \mathsf{OKVS.Decode}(D, (i, [\![c^{(i)}]\!]_N^R)) + f^{(i)} + H_\phi(i)$, where $i = i_j^R$,

and from step 3 of Π_{SpSOT}, the $H_\phi(i)$ is pseudorandom to S. This directly implies that g_j^R is pseudorandom to S for every $i_j^R \in I^R \setminus I^S$, and concludes our proof for Equation (6).

□

D.2 Security for Sp-daROT

Theorem 4. *Let Π be a secure 0-round da-ROT protocol in the $\mathcal{F}_{SOT}$ hybrid model. Then sparse-comp(Π) in Figure 4 is a secure Sp-daROT protocol for the same parameters in the $\mathcal{F}_{Sp\text{-}daROT}$ hybrid model.*

Proof. The correctness of the Sp-daROT is easy to verify. The key observation is that the output of each invocation of SpSOT can be replaced by SOT outputs for matching indexes, and with random elements otherwise. More formally, for any $(i, x) \in X'$, the receiver's input to $\mathcal{F}_{\mathsf{SpSOT}}^{\mathsf{param}}[j]$, the corresponding output O_R contains (i, r) defined as follows:

- If there exists $(i, y) \in Y'$ for some y: Add the element (i, r) to O_R where $(r, *) \leftarrow \mathcal{F}_{\mathsf{SOT}}^{\mathsf{param}[j]}(x, y)$
- Otherwise: Add (i, r) to O_R, where r is a random value.

Therefore, the correctness of Sp-daROT follows directly from the correctness of the underlying da-ROT protocol Π.

To complete the privacy proof, we construct a simulator for Sp-daROT in a natural way: For a corrupt sender with input X and output X_{out}, we run the da-ROT simulator $\mathsf{Sim}((i, X[i]), X_{\mathsf{out}}[i])$ for each index i in X. This outputs a distribution indistinguishable from the multiple SOT calls in the underlying da-ROT protocol. Hence, by construction, concatenating the distributions of $\mathsf{Sim}((i, X[i]), X_{\mathsf{out}}[i])$ for each index in the input set gives the view of the sender in Sp-daROT. A similar argument follows for a corrupt receiver.

□

D.3 Security for fuzzyPSI

Theorem 5. *The protocol in Figure 5 securely implements the $\mathcal{F}_{\mathsf{fuzzyPSI}}$ functionality in the $\mathcal{F}_{Sp\text{-}daROT}$ hybrid model.*

Proof. **Correctness:** For any $b \in B$, we consider two cases:

- There exist $a \in A$ such that $\mathtt{dist}(a, b) \le \delta$: By correctness of spatial hashing, Bob's dictionary B' contains a key-value pair $((\mathsf{cell}(a), j), b)$ for some $j \in [\rho_\eta^A]$. By the correctness of $\mathcal{F}_{\mathsf{Sp\text{-}daROT}}$ invoked in Step 2, Alice and Bob's out dictionaries, respectively, contain $((\mathsf{cell}(a), j), (r_0, r_1))$ and $((\mathsf{cell}(a), j), r_0)$ for some r_0, r_1. Hence, set Y contains $((\mathsf{cell}(a), j), \mathsf{Enc}(r_0, b))$. In Step 5 for the loop for the $((\mathsf{cell}(a), j), r_o)$ iteration, Alice computes $\mathsf{Dec}(r_0, c)$ and adds it to Z. This, by the correctness of the encryption scheme, is precisely b.
- For all $a \in A$, $\mathtt{dist}(a, b) > \delta$: we complete this proof by showing $b \notin Z$. Assume by contradiction $b \in Z$, which is the output of Alice. By the correctness of one-time pad encryption, OKVS D sent by Bob contains a key-value pair $(i, \mathsf{Enc}(r, b))$ for some i, r where Alice's set A_{out} contains (i, r). By the correctness of $\mathcal{F}_{\mathsf{Sp\text{-}daROT}}$, Alice's input set A' contains (i, a') for some $a' \in A$ such that $\mathtt{dist}(a, b) \le \delta$ with all but negligible probability. This gives us a contradiction since, as assumed, no element in A was δ close to b.

Privacy: We divide this proof into two cases:

- **Corrupt Bob**: Bob's view in the protocol only contains B_{out} the output of $\mathcal{F}_{\mathsf{Sp\text{-}daROT}}$ from Step 3, an OKVS of fixed size. Hence, by the OKVS obliviousness property, this OKVS is indistinguishable from another OKVS with key-value pairs with arbitrary keys and corresponding random values, which is simulatable given public parameters.
- **Corrupt Alice**: Alice view contains A_{out} the output of $\mathcal{F}_{\mathsf{Sp\text{-}daROT}}$ from Step 2 and an OKVS D from Step 4. We first construct a hybrid where the OKVS D constructed in Step 3 of the protocol is modified as follows: for each $(i, \mathsf{Enc}(r_0, b))$ inserted in Y where $b \notin A \cap B$, we replace it by (i', r) where r is a random string of same length as $\mathsf{Enc}(r_0, b)$ and i' is a random index. This hybrid has a computationally indistinguishable distribution compared to the original protocol since the ciphertexts are pseudorandom for elements not

in the intersection, and OKVS satisfies the independence property.

As the final hybrid, we replace the output of $\mathcal{F}_{\text{Sp-daROT}}$ from Step 2 (i.e. A_{out}) with another OKVS with random key-value pairs for key indexes not corresponding to the intersection.

□

Theorem 6. *The protocol in Figure 5 securely implements the $\mathcal{F}_{\text{fuzzyCard}}$ and $\mathcal{F}_{\text{fuzzyJoin}}$ functionality in the $\mathcal{F}_{\text{Sp-daROT}}$, $\mathcal{F}_{\text{PSICard}}$, $\mathcal{F}_{\text{PSIJoin}}$ hybrid model.*

We omit the proof of Theorem 6, as it follows the same blueprint as that of Theorem 5.

E PSI IDEAL FUNCTIONALITIES

PSI Ideal Functionalities.

Parameters: Input set sizes n, m, input length ℓ, $\mathcal{D} = \{0,1\}^*$.

- The two parties, receiver Alice and sender Bob, provide sets $A, B \subseteq \mathcal{D}$ as input, respectively, with $n = |A|$ and $m = |B|$.
- Only for $\mathcal{F}_{\text{PSIJoin}}$: the sender and receiver also input associated data dictionaries AD^S, AD^R respectively with key sets A, B and values in $\{0,1\}^\ell$,
- Define outputs for each functionality as follows:
  - $\mathcal{F}_{\text{PSICard}}$: output $|A \cap B|$ to the receiver Alice.
  - $\mathcal{F}_{\text{PSIJoin}}$:
    - Initialize $t = 0$.
    - For every $(a, b) \in A \times B$ where $a = b$:
      - sample $u_t \leftarrow_\$ \{0,1\}^{2\sigma}$
      - set v_t such that $u_t \oplus v_t = \text{AD}^S(a)||\text{AD}^R(b)$
      - $t \leftarrow t + 1$
  - Shuffle both $\vec{u}$ and $\vec{v}$ with the same random permutation
  - Output vectors $\vec{u}$ to sender and $\vec{v}$ to receiver

F RELATED WORK

In this section, we review advancements in malicious fuzzy PSI protocols. We begin with the work of Garimella et al. [14], which introduced the first specialized fuzzy PSI protocols for the L_1 and L_∞ distance metrics in 2022. Their approach introduced novel constructions, such as weak FSS, and innovative techniques like spatial hashing to reduce the communication complexity of these protocols. Among these contributions, spatial hashing stands out as a significant innovation and has been adopted in numerous subsequent fuzzy PSI works.

Building on this foundation, [37] relies on the DDH assumption to propose new spatial hashing techniques for a fuzzy PSI protocol that supports metrics $L_{\text{p}\in[1,\infty]}$. By leveraging the homomorphic properties of DDH tuples, they enabled efficient evaluation of comparison functions across various distance metrics. Additionally, their spatial hashing techniques exploit the geometric structure of the space, avoiding the quadratic blowup typically encountered when comparing all receiver balls to all sender points. However, their protocol relies heavily on public-key operations.

Gao et al.[9] observed that most fuzzy PSI protocols adhere to a two-phase paradigm: coarse mapping and refined filtering. The coarse mapping phase identifies receiver and sender points that are "close enough," while the refined filtering phase performs fuzzy matching between these paired points. The paper introduced a new interactive primitive called fuzz-mapping, which abstracts the coarse mapping phase. By combining fuzzy-mapping with public-key cryptography, they developed new fuzzy PSI protocols that support L_∞, L_p, and Hamming metrics. However, a major drawback to this work is their reliance on a different and unrealistic disjointness assumption, where each sender's or receiver's point is separated from other points in the same set by at least one dimension.

In the most recent work considered in this section, Richardson et al. [31] proposed a general fuzzy-psi protocol framework, which they use to instantiate protocols for L_p and L_∞ metrics in the same work. The starting point for the framework is the PSI protocol presented in [6], which compiles a private equality test (PEQT) protocol into a PSI protocol. [31] generalizes this protocol to the fuzzy PSI setting by combining it with garble circuit and spatial hashing techniques, supporting L_p and L_∞ metric comparisons. Different garbling schemes are recommended depending on the metric to achieve better performance when instantiating their framework.

[36] presents fuzzy PSI protocols for Hamming distance, while [4] extends this to support both Hamming distance and the L_1 metric. However, we do not consider these works in our comparisons due to some major limitations of their protocols. More specifically, their protocol [4] for hamming distance has a non-negligible false positive rate, and their protocol for L_1 metric only supports one-dimensional spaces.

A sparse matching functionality was presented in [35] termed Read-Only Oblivious Maps (ROOMs). In this primitive, the server holds a dictionary D, and the client holds a set of key values $\vec{k}$. As output the parties hold random secret sharing of vector $\vec{v}$ such that $\vec{v}_i$ is some default value β if $\vec{k}_i$ is not in D, else $\vec{v}_i = D[\vec{k}_i]$. This can be viewed as a special case of our sparse da-ROT primitive but only for exact matches. Our work focuses on the fuzzy setting, and hence, we provide no direct comparison with their constructions.

We note that our proposed da-ROT primitive shares a definition similar to conditional OT (SCOT) [2] and membership OT [7]. While membership OT can be viewed as a special case of da-ROT that focuses solely on equality, previous works on SCOT are restricted to one-dimensional, interval-based conditions (i.e., L_∞ in 1D) and depends on computationally expensive homomorphic encryption. In contrast, da-ROT supports a wider range of distance metrics and leverages standard OT. Combined with our sparse compiler, this enables efficient implementations of sparse and batched da-ROT via OT extensions. We believe this advancement will be particularly valuable to the applied MPC literature, where SCOT is used as a building block.