Privacy-preserving Proximity Testing from Geometric Fuzzy Matching

Ioannis Katis
University of St. Gallen
St. Gallen, Switzerland
ioannis.katis@unisg.ch

Katerina Mitrokotsa
University of St. Gallen
St. Gallen, Switzerland
katerina.mitrokotsa@unisg.ch

Florias Papadopoulos
University of St. Gallen
St. Gallen, Switzerland
florias.papadopoulos@unisg.ch

Abstract

Proximity testing is crucial to location-privacy applications, from discovering nearby friends to enabling UAV collision avoidance. In such settings, users must determine proximity without revealing their exact locations. This motivates privacy-preserving proximity testing (PPPT) protocols revealing only if the proximity condition holds, while hiding both parties' inputs. However, most existing PPPT protocols rely on strong assumptions (e.g., non-colluding servers) or require simultaneous interaction, limiting their practicality. Moreover, they typically define proximity using metric distances (e.g., Euclidean distance), failing to support richer membership queries for complex regions like buildings or parks.

To address these, we introduce a new primitive called *Geometric Fuzzy Matching* (GFM), which generalizes fuzzy matching to arbitrary n-dimensional regions. In GFM, the receiver specifies a region and learns only whether the sender's location lies within it, without revealing either party's input. This approach captures both classical distance-based proximity checks (for any Minkowski ℓ_p norm, $1 \leq p \leq \infty$), as well as membership tests for complex regions, providing a unified framework for diverse proximity queries.

In low-dimensional settings, our protocol improves on distance-based checks compared to state-of-the-art van Baarsen *et al.* (EUROCRYPT 2024) for ℓ_∞ and maintains stable practical efficiency for ℓ_p norms under large distance thresholds or for $p \geq 4$, where previous approaches quickly become computationally prohibitive. It is also the *first* to support fuzzy matching over arbitrary geometric regions, enabling proximity queries in complex spaces. Our implementation confirms these results and demonstrates the protocol's efficiency and applicability across diverse PPPT scenarios.

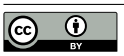
CCS Concepts

• **Security and privacy → Cryptography**.

Keywords

Fuzzy Private Set Intersection, Fuzzy Matching, Proximity Tracking

ACM Reference Format:
Ioannis Katis, Katerina Mitrokotsa, and Florias Papadopoulos. 2026. Privacy-preserving Proximity Testing from Geometric Fuzzy Matching. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '26), June 1–5, 2026, Bangalore, India.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3779208.3785374



This work is licensed under a Creative Commons Attribution 4.0 International License.
ASIA CCS '26, Bangalore, India
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2356-8/26/06
https://doi.org/10.1145/3779208.3785374

1 Introduction

Privacy-preserving proximity testing (PPPT) is a critical requirement in location-based services (LBS), where individuals, points of interest, and services must be discoverable without compromising user privacy. Applications like ride-sharing platforms (e.g., Uber) have faced significant criticism for violating users' location privacy, highlighting the need for privacy-preserving mechanisms in LBS.

However, existing PPPT protocols frequently depend on impractical assumptions, such as the non-collusion of servers [23, 24] or the constant online availability of both parties during proximity checks through requiring multiple rounds of interaction [13, 30]. These constraints make such protocols difficult to deploy in practice, as they are inefficient even in one-to-one scenarios and become entirely impractical when scaled to larger settings. Consequently, companies have not adopted private solutions in practice, particularly for large-scale one-to-many settings like ride-sharing, where centralized handling of location data remains the norm despite the lack of strong privacy guarantees. This is exemplified by Uber's €290 million fine in 2024 [14] for transferring European drivers' location data to U.S. servers without adequate GDPR safeguards, highlighting privacy challenges in the industry.

Another key limitation lies in the reliance on distance metrics. Most current approaches [5, 19, 22, 27, 30] assess proximity by checking if two entities are within a specified threshold distance. This model fails in scenarios where proximity is better defined by spatial regions rather than distances, e.g., when identifying if two users are within the same campus, building, or even the same floor.

To address these challenges, *fuzzy matching* emerges as a promising approach. Fuzzy matching was first introduced by Freedman *et al.* [4] in the context of identifying whether two binary strings differ in fewer than a threshold number of positions (i.e., under Hamming distance). However, in recent years the concept has been extended to support general metric spaces [6, 29], particularly focusing on ℓ_p distances for $p \in [1, \infty]$. In its current formulation, it is a protocol that takes as input a point $\boldsymbol{w}$ from the receiver and a point $\boldsymbol{q}$ from the sender, both in the space $\mathbb{Z}^d$. The protocol returns 1 to the receiver if the distance between $\boldsymbol{w}$ and $\boldsymbol{q}$ is less than or equal to a threshold δ, and 0 otherwise. However, despite its promise, fuzzy matching has so far been limited to metric-based proximity (e.g., ℓ_p distances), and does not yet support region-based proximity definitions required in many real-world applications.

In this work, we extend fuzzy matching to the geometric setting, introducing a new abstraction, *geometric fuzzy matching*, that enables both private distance comparisons and private region-membership tests. This allows us to answer positively:
Can we design an efficient, privacy-preserving protocol that supports both distance- and region-based proximity checks?

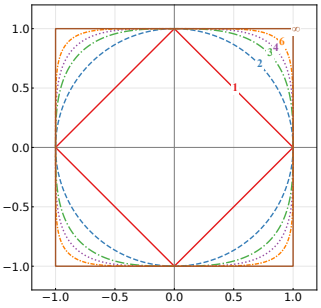


Figure 1: Visualization of ℓ_p norm unit balls.

1.1 Contributions

We begin by improving existing fuzzy matching methods through a more efficient protocol for the ℓ_∞ norm in low-dimensions, a setting well suited for location-based applications. Our construction reduces what would otherwise scale linearly with the distance threshold δ to near-logarithmic complexity, significantly lowering computational and communication costs compared to van Baarsen *et al.* [29], and enabling the handling of much larger δ in practice.

Next, we aim to improve protocols for generic ℓ_p distances, used in settings such as grid-based routing (ℓ_1), Euclidean proximity (ℓ_2), and deviation-sensitive applications requiring higher-order norms. Existing approaches quickly become computationally impractical even for small values of δ once p increases. To overcome this, we adopt a more geometric perspective for fuzzy matching: receiver has a d-dimensional point with a surrounding d-dimensional region determined by a distance threshold δ, and the goal is to check whether the sender's point lies within this region. This area is a hyperrectangle[1] when the ℓ_∞ distance is used, a hyperball for ℓ_2, and an ℓ_p-ball in general (see Figure 1). By decomposing these regions into hyperrectangles and residual points, we can apply our improved ℓ_∞ protocol to all hyperrectangles concurrently, efficiently handling the entire region. This not only makes fuzzy matching for ℓ_p norms practical for all $p \geq 2$, but also extends naturally beyond ℓ_p balls to arbitrary, irregular regions, which can likewise be partitioned into hyperrectangles. From this observation, we arrived at the definition of a new cryptographic primitive, *Geometric Fuzzy Matching* (GFM), which unifies these ideas and enables both private distance comparisons and private region-membership tests.

We demonstrate the practical impact of our contributions by applying GFM to privacy-preserving proximity testing. Our framework enables a receiver to specify arbitrary spatial regions—such as a floor plan or a 3D building layout—and privately determine whether a sender's location lies within that region, without either party revealing additional information. Unlike prior protocols that require trusted third parties or server-based assumptions [23], our solution operates in a fully two-party model without trusted servers, enhancing deployability and privacy guarantees. Additionally, our framework can handle arbitrary, irregular regions, whereas previous protocols are restricted to distance-based proximity tracking and cannot accurately support complex spatial areas.

Putting everything together, our key contributions are:

(i) We construct an efficient fuzzy matching protocol for the ℓ_∞ norm in low-dimensional spaces, reducing complexity from linear in the distance threshold δ to near-logarithmic, enabling practical handling of larger thresholds.
(ii) We introduce *Geometric Fuzzy Matching* (GFM), a new cryptographic primitive that generalizes fuzzy matching to arbitrary regions, supporting both private distance comparisons and private region-membership tests. In the distance comparison setting, GFM can maintain its efficiency for ℓ_p distances even for large p, where prior protocols become computationally impractical.
(iii) We develop novel partitioning algorithms that approximate d-dimensional regions via axis-aligned hyperrectangles, enabling scalable and flexible instantiations of GFM.
(iv) We apply GFM to privacy-preserving proximity testing, enabling receivers to specify complex spatial regions—such as floor plans or 3D layouts—and privately determine whether a sender's point lies within them without revealing additional information.
(v) We demonstrate our protocols' efficiency through experimental evaluation, and comparison with the two-message fuzzy matching protocols of van Baarsen *et al.* [29].

1.2 Technical Overview

To support rich, asynchronous location-privacy applications, we develop a two-message fuzzy matching (FM) protocol for arbitrary geometric regions without relying on trusted infrastructure. Existing protocols, such as those by Gao *et al.* [6] and van Baarsen *et al.* [29], are distance-based and do not directly generalize to arbitrary regions. Moreover, while the work of Gao *et al.* offers an efficient online phase, offline preprocessing with communication is also required, which is incompatible with asynchronous settings; for the same reason, structure-aware PSI protocols [7, 9, 10] are also unsuitable. Thus, we base our cryptographic foundation on the protocol of van Baarsen *et al.* [29], which supports two-message execution and no prior communication.

FM Protocol of van Baarsen *et al.* [29]. In the simplest case of ℓ_∞ distance, the protocol enables two parties to test whether the distance between their inputs is at most a given threshold, using additively homomorphic encryption (AHE) and Oblivious Key-Value Stores (OKVS)[2]. In their construction, the receiver encodes, for each dimension, all values within distance δ of its input into an OKVS filled with encryptions of zero, and sends these encodings together with the AHE public key to the sender. The sender queries at its coordinates, homomorphically sums the obtained ciphertexts, and re-randomizes the result before returning it. The receiver then decrypts and learns whether the distance condition holds. Correctness follows from the fact that all queried ciphertexts encrypt zero if and only if the points are within δ in every coordinate, while security is ensured by the obliviousness of the OKVS against the sender and by ciphertext re-randomization against the receiver.

Improved Fuzzy Matching for Infinity Distance. As a first step toward realizing GFM, we improve the efficiency of the previous protocol by utilising the Distance-Aware Set Augmentation (DASA) technique of Chakraborti *et al.* [3], which represents an interval using a minimal collection of binary prefixes instead of listing each element individually. This prefix-based representation yields

[1] A hyperrectangle is the space defined by the cartesian product of ranges (equivalent to a rectangle in two dimensions).

[2] OKVS are data structures that encode key-value pairs while hiding the underlying key set.

a more compact OKVS, significantly reducing communication and computational effort on the receiver's side. The trade-off is that the sender must now evaluate $O(\log \delta)$ prefixes per dimension, resulting in a communication increase on the sender. However, this overhead remains acceptable for low-dimensional settings.

Geometric Fuzzy Matching. Having improved the ℓ_∞ protocol, we next ask how to support the other ℓ_p distances. Here it is useful to adopt a geometric viewpoint: testing whether two points are within distance δ under some ℓ_p norm is equivalent to testing whether the sender's input lies inside a geometric region (an ℓ_p-ball) defined around the receiver's input. This shift in perspective immediately suggests a broader goal—supporting membership tests for arbitrary geometric regions, not only ℓ_p balls. We call the resulting framework Geometric Fuzzy Matching (GFM).

Our construction builds on the observation that any bounded, discrete region in $\mathbb{Z}^d$ can be efficiently represented by a collection of disjoint axis-aligned hyperrectangles together with a minimal set of boundary points. This representation is obtained by partitioning the region through a core technical primitive, the Partitioner (Section 3). Hyperrectangles are then handled independently using instances of the improved ℓ_∞ protocol, while the boundary points are checked individually. All instances execute in parallel, and their outputs are aggregated into a single membership bit. This approach provides a general, modular framework for GFM.

2 Preliminaries

Notation. We represent the computational security parameter as $\lambda \in \mathbb{N}$ and the statistical security parameter as $\kappa \in \mathbb{N}$. We denote by $\mathbb{Z}_q$ the finite field of integers modulo a prime q. We write $[n]$ for the set $\{1, \ldots, n\}$, where $n \in \mathbb{N}$ and $[a : b]$ for the set $\{a, a+1, \ldots, b\}$, where $a, b \in \mathbb{Z}$ with $a < b$. We denote vectors $\boldsymbol{x} = (x_1, \ldots, x_d)$ in bold. We denote with $a \leftarrow\!\!\$\ A$ the fact that a is picked uniformly from a set A. Finally, computational and statistical indistinguishability are denoted by $\approx_c$ and $\approx_s$, respectively.

Parties. We consider two mutually distrustful parties, a receiver and a sender, operating in the semi-honest model (i.e., they do not deviate from the protocol but may attempt to infer additional information). The protocols are designed for this two-party setting and analyzed via the standard simulation-based paradigm [15].

2.1 Distance Aware Set Augmentation (DASA)

Distance-Aware Set Augmentation (DASA) is a technique introduced by Chakraborti *et al.* [3] to reduce the communication overhead of their Distance-Aware Private Set Intersection (DA-PSI) protocol for the Minkowski distance of order 1 over integers. Given an input $a \in \mathbb{Z}_{\geq 0}$ and an integer threshold $\delta > 0$, DASA aims to compactly encode the range $[a-\delta : a+\delta]$ using a small set of binary strings with wildcard bits.[3] Instead of listing all $2\delta + 1$ integers in this interval, it captures them via binary prefixes that generalize to entire subtrees in a prefix tree.

In more detail, DASA constructs a prefix tree over the MaxBitLen-bit representations of the neighborhood and identifies all maximal enclosing complete subtrees—that is, the largest complete binary subtrees whose leaves lie entirely within the interval. Each such subtree corresponds to a shared prefix which can be extended with wildcard bits to represent all its leaves. These wildcard-extended prefixes form the receiver's augmented set. For the sender, a similar set of strings is derived from his input point $b \in \mathbb{Z}$ such that a match with any receiver string implies $|a - b| \leq \delta$. The full procedure is described in Algorithm 1.

[3]The original results apply to open intervals; we adjust our algorithm and theorem for closed intervals accordingly.

Algorithm 1 Distance-Aware Set Augmentation

Input: Receiver's input $a \in \mathbb{Z}$, sender's input $b \in \mathbb{Z}$ and $\delta \in \mathbb{N}$
Output: Augmented sets A and B

procedure $\mathsf{DASA}_{\mathcal{R}}(a, \delta)$
  Initialize empty set A
  for each integer $a_i \in [a - \delta : a + \delta]$ **do**
    $s_i \coloneqq$ Bit string of MaxBitLen-bits representing a_i
  end for
  Construct $P \coloneqq$ prefix trie with leaves $\{s_1, \ldots, s_{2\delta+1}\}$
  for each maximal enclosing complete subtrie T in P **do**
    $\varphi \coloneqq$ prefix of T in P
    Append "don't care" bits ($*$) to φ up to MaxBitLen
    Add φ to A
  end for
  return A
end procedure

procedure $\mathsf{DASA}_{\mathcal{S}}(b, \delta)$
  Initialize empty set B
  $s \coloneqq$ Bit string of MaxBitLen representing b
  for $i = 0, 1, \ldots, \lfloor \log(2\delta + 1) \rfloor$ **do**
    $s' \coloneqq s[\mathsf{MaxBitLen} - 1] \ldots s[i] * \cdots *$
    Add s' to B
  end for
  return B
end procedure

Number of Augmented Elements. Results from [3] establish the following bounds on the number of representative strings:

Theorem 1. For any $a, b \in \mathbb{Z}$ and integer threshold $\delta > 0$, the receiver's augmented set A output by $\mathsf{DASA}_{\mathcal{R}}(a, \delta)$ contains $O(\log \delta)$ strings, and the sender's augmented set $\mathsf{DASA}_{\mathcal{S}}(b, \delta)$ contains exactly $\lfloor \log(2\delta + 1) \rfloor + 1 = O(\log \delta)$ strings.

However, the exact count of the receiver's strings can vary with the point a and threshold δ, potentially leaking information. To mitigate this, the augmented set output by $\mathsf{DASA}_{\mathcal{R}}$ should be padded to the maximum possible number for given δ, regardless of a. It is easy to prove that this maximum number of strings is bounded by

$$2\lfloor \log(2\delta + 1) \rfloor.$$

This follows from the fact that the strings represent dyadic intervals of lengths that are powers of two covering the radius-δ neighborhood around a. At each level k of the full binary tree, for $0 \leq k \leq \lfloor \log(2\delta+1) \rfloor$, there are at most two such intervals, yielding the stated bound.

2.2 Oblivious Key-Value Store (OKVS)

Polynomial encodings, dating back to at least [18], hide a set of distinct input keys $\{k_1, \ldots, k_n\}$ by interpolating a polynomial P through points (k_i, v_i) such that $P(k_i) = v_i$, where random values v_i effectively conceal the keys k_i. Oblivious Key-Value Stores (OKVS), introduced by Garimella *et al.* [8], generalize this concept as data

structures that encode key-value pairs while hiding the underlying key set. Subsequent works improved on their result, proposing new constructions [2, 26] that achieve efficiency suitable for practical PSI [16, 26] and fuzzy PSI [7, 9, 29] applications. Further information is provided in Appendix A.

2.3 Random Self-Reductions of DDH Tuples

The security of the FM protocols in [29] relies on the random self-reduction of decisional Diffie-Hellman (DDH) tuples, a technique originally introduced by Naor and Reingold [21] and first applied to PSI in Apple's PSI protocol [1]. This self-reduction enables re-randomizing DDH tuples securely, thus allowing a receiver to learn if there is a match, without revealing additional information.

This builds on the decisional Diffie-Hellman (DDH) assumption, which states that, for a cyclic group $\mathbb{G} = \langle g \rangle$ of prime order q, the distribution of DH tuples $(g, h \coloneqq g^s, h_1, h_1^s)$, where $s \leftarrow\$ \mathbb{Z}_q$ and $h_1 \leftarrow\$ \mathbb{G}$, is computationally indistinguishable from the one of random tuples $(g, h \coloneqq g^s, h_1, h_2)$, where $s \leftarrow\$ \mathbb{Z}_q$ and $h_1, h_2 \leftarrow\$ \mathbb{G}$.

Naor and Reingold show that determining if a tuple (g, h, h_1, h_2), with $h, h_1, h_2 \in \mathbb{G}$ is a DH tuple reduces to solving DDH. In the FM setting of [29], they focused on a special case of this reduction where $h = g^s$ is fixed:

Lemma 1 (Random Self-Reduction [21]). Let $\mathbb{G} \coloneqq \langle g \rangle$ be a cyclic group of order q, $h \coloneqq g^s$ for $s \in \mathbb{Z}_q$ and $h_1, h_2 \in \mathbb{G}$. If $h_1' \coloneqq g^a h_1^b$ and $h_2' \coloneqq h^a h_2^b$, where $a, b \xleftarrow{\$} \mathbb{Z}_q$, then:

- if $h_2 = h_1^s$, h_1' is uniformly random in $\mathbb{G}$ and $h_2' = (h_1')^s$,
- otherwise, (h_1', h_2') is a uniformly random pair of group elements.

2.4 Fuzzy Matching Functionalities

Many proximity-testing protocols rely on predefined distance metrics to determine closeness between input points. For vectors $\boldsymbol{w}, \boldsymbol{q} \in \mathbb{Z}^d$, common metrics include:

- the ℓ_∞ distance: $\mathrm{dist}_\infty(\boldsymbol{w}, \boldsymbol{q}) = \|\boldsymbol{w} - \boldsymbol{q}\|_\infty = \max_{i \in [d]} |w_i - q_i|$,
- the ℓ_p distance $\mathrm{dist}_p(\boldsymbol{w}, \boldsymbol{q}) = \|\boldsymbol{w} - \boldsymbol{q}\|_p = \left(\sum_{i=1}^d |w_i - q_i|^p\right)^{1/p}$.

Such settings could be addressed by the fuzzy matching functionality $\mathcal{F}_{\mathrm{FM}}$ from [29], which formalizes this as a distance comparison between two points, parameterized by a fixed distance function and threshold. This formulation is well-suited to symmetric ($\mathrm{dist}(\boldsymbol{w}, \boldsymbol{q}) = \mathrm{dist}(\boldsymbol{q}, \boldsymbol{w})$), metric-based proximity tests, effectively treating the receiver's input as a fixed-radius neighborhood around a point. For example, under ℓ_∞ in dimension $d = 3$, the receiver's input corresponds to all points in $\mathbb{Z}^d$ inside a hypercube.

However, this functionality requires the structure of Alice's input to be fixed in advance through the parameters. In our applications, we want Alice to provide inputs corresponding not only to asymmetric regions but also to different, potentially varying geometric regions in each run of the protocol, without being restricted to a predetermined parametrization.

To overcome this limitation, we extend FM to a more general notion of geometric fuzzy matching (GFM), formalized via the functionality $\mathcal{F}_{\mathrm{GFM}}$. This functionality generalizes fuzzy matching to arbitrary region-based predicates. In this setting, the receiver specifies an acceptance region either explicitly as a subset $\mathcal{R}_{\mathsf{input}}$ of a universe $\mathcal{U}$ or concisely via a structured description $\mathcal{R}_{\mathsf{desc}}$. The functionality then checks if the sender's point lies in this region.

Ideal Functionality of [29] for Fuzzy Matching ($\mathcal{F}_{\mathrm{FM}}$)

Parameters: dimension d, threshold δ, distance function $\mathrm{dist}(\cdot, \cdot)$.
Functionality:
- Receiver inputs $\boldsymbol{w} \in \mathbb{Z}^d$.
- Sender inputs $\boldsymbol{q} \in \mathbb{Z}^d$.
- Output 1 to Receiver if $\mathrm{dist}(\boldsymbol{w}, \boldsymbol{q}) \leq \delta$, and 0 otherwise.

However, in a setting where no a priori information is available about the structure of the receiver's input—beyond the fact that it lies within some known universe $\mathcal{U}$—the sender requires a description of this structure in order to operate correctly. In the real protocol, this information is included as additional data in the receiver's first message. In the ideal world, we model this via a new ideal functionality $\mathcal{F}_{\mathrm{Info}}$, which takes as input a description of the receiver's set and outputs structural information about the region (i.e., how the region is shaped, without revealing its position in the space). This functionality is invoked by $\mathcal{F}_{\mathrm{GFM}}$, and provides the necessary metadata for checking whether the sender's point lies in the receiver's region $\mathcal{R}_{\mathsf{input}} \subseteq \mathcal{U}$.

Ideal Functionality for Geometric FM ($\mathcal{F}_{\mathrm{GFM}}$)

Parameter: A universe $\mathcal{U} = (\{0, \ldots, 2^{\mathsf{MaxBitLen}-1}\}^d)$
Functionality:
- Receiver inputs $\mathcal{R}_{\mathsf{input}} \subseteq \mathcal{U}$ (or a concise description $\mathcal{R}_{\mathsf{desc}}$ of $\mathcal{R}_{\mathsf{input}}$)
- Sender inputs $\boldsymbol{q} \in \mathbb{Z}^d$.
- $\mathcal{F}_{\mathrm{Info}}$ is invoked, and R_{info} is output to both parties.
- Output 1 to Receiver if $\boldsymbol{q} \in \mathcal{R}_{\mathsf{input}}$, and 0 otherwise.

Ideal Functionality for Region Info Extraction ($\mathcal{F}_{\mathrm{Info}}$)

Parameter: A universe $\mathcal{U} = (\{0, \ldots, 2^{\mathsf{MaxBitLen}} - 1\}^d)$
Functionality:
- Receiver inputs $\mathcal{R}_{\mathsf{input}} \subseteq \mathcal{U}$ (or a concise description $\mathcal{R}_{\mathsf{desc}}$ of $\mathcal{R}_{\mathsf{input}}$)
- Outputs R_{info}.

3 Geometric Partitioning Algorithms

Extending Fuzzy Matching requires encoding the receiver's input as collections of axis-aligned hyperrectangles, enabling their efficient representation as strings via Distance-Aware Set Augmentation. To achieve this, we propose a geometric partitioning approach that discretizes d-dimensional surfaces into unions of non-overlapping hyperrectangles and residual lattice points—points inside the surface but outside these hyperrectangles. We offer a general scheme for such partitioning, complemented by optimized variations designed for different applications and resource constraints.

3.1 Partitioning a d-Dimensional Surface

This subsection presents our method for partitioning a closed and compact d-dimensional surface into a manageable collection of axis-aligned hyperrectangles. The goal is to approximate the surface

efficiently while exactly capturing all integer lattice points within it. The resulting partition enables a compact representation of these integer points, controlled by a resolution parameter that trades off coverage tightness against complexity.

To achieve this, we introduce a partitioning subroutine, CutAndFuse, consisting of two key steps: cutting and fusing. In the *cutting step*, the minimal axis-aligned bounding box of S is uniformly divided into a grid of axis-aligned hypercubes, where the side length of each cube is equal to a *resolution* parameter r. From those cells, we retain only those that are entirely contained within S, forming an initial partition of a subset $S' \subseteq S$. Next, the *fusing step* iteratively merges adjacent cells that have a common $(d-1)$-dimensional boundary and align along all other dimensions. This fusion significantly reduces the number of components, yielding a more concise partition of S'. Additionally, we note that, since the fused partition only covers S', the remaining region $S \setminus S'$, which may still contain lattice points, is handled in a subsequent step.

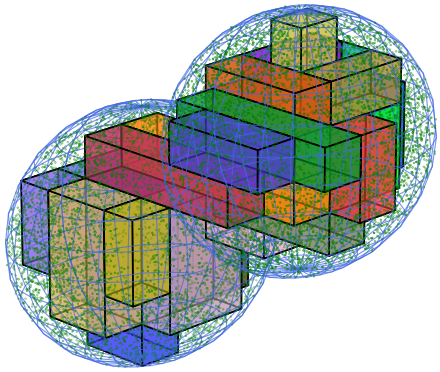


Figure 2: Illustration of Partitioner **on input the union of two spheres (centers: $c_1 = (15, 26, 42)$, $c_2 = (50, 42, 17)$; radius: $R_1 = R_2 = 30$). Resolution $r = 10$ minimizes a cost function balancing sender and receiver computation in GFM. A random subset of the total residual points are denoted with green.**

Building on this subroutine, the complete approach is formalized in the Partitioner$(\cdot)$ algorithm (Figure 3). The procedure explores a range of possible candidate resolutions r, invoking CutAndFuse at each one. For every such resolution, it computes a cost function $\mathrm{Cost}(r)$, determined by our overall framework, which reflects trade-offs in the resulting partition. The resolution r^* minimizing this cost is selected as optimal. Finally, the algorithm computes and outputs the fused hyperrectangles and residual points for the chosen r^*.

On the Cost **function.** The cost function $\mathrm{Cost}(r)$ captures trade-offs in partitioning the surface S at resolution r, balancing factors such as the number of fused hyperrectangles N_r, their total volume V_r, and the number of uncovered lattice points L_r. In our setting, these quantities correspond to concrete sources of computational overhead, such as increased looping, memory access, or function calls. For instance, a large N_r could translate to increased work for the sender due to more function invocations.

Our priority in the overall framework is to manage computational complexity for both sender and the receiver. Consequently, the cost function we use there quantifies and balances these costs, with its precise formulation detailed in the following section.

Selecting the optimal resolution r^*. The procedure for finding the optimal resolution r^* depends on the behavior of the cost function $\mathrm{Cost}(r)$. Since $\mathrm{Cost}(r)$ is not guaranteed to be monotonic or convex, an exhaustive search over a discrete set of candidate resolutions is often required to ensure that the global minimum is found. However, such a strategy can become computationally expensive, especially for very large surfaces. In practice, to improve scalability, one may resort to more efficient searching strategies such as adaptive refinement (i.e., finer search near promising regions), or heuristic methods (e.g., Bayesian optimization [20]), which offer faster convergence at the cost of exact optimality.

Remark 1. We note that the Partitioner procedure could be extended to handle multiple surfaces, including those with holes or more complex geometries. This would allow our overall framework to support more general input regions, effectively enabling a form of fuzzy matching when Alice's/receiver's input covers a broader or more complicated area. However, this added flexibility may reduce efficiency due to increased computational complexity. In such cases, more optimized partitioning algorithms that exploit the geometric structure of Alice's input would likely be preferred.

Efficiency. In most applications of GFM, partitioning can be computed beforehand and may even be stored in a database for common shapes, such as spheres of various radii. Consequently, we treat geometric partitioning as a preprocessing step and do not analyze its efficiency further in this work. Nevertheless, for completeness, we provide example runtimes of our partitioning algorithm on balls of different ℓ_p distances ($p = 2, 5, 10$) and thresholds δ: in 2D, runtimes range from 4–5 ms for $\delta = 30$ to 58–60 ms for $\delta = 100$, and 15–24 s for $\delta = 1000$; in 3D, from 366 ms (ℓ_{10}) to 950 ms (ℓ_2) for $\delta = 30$, and 59–322 s for $\delta = 100$.

3.2 Optimized Partitioning

Although Partitioner is versatile, it does not exploit structural properties inherent to specific surfaces or scenarios. For these instances, alternative algorithms can be used. Examples of customized partitioning algorithm include those that partition surfaces like discs, rectilinear polygons (see Appendix B.1), and unions of axis-aligned hyperrectangles, with improved computational efficiency.

3.2.1 Partitioning discs. We extend Andrew Glassner's "square-packing in a circle" method [11] by first generating a packing of axis-aligned squares inside the disc and then identifying the largest lattice-aligned squares contained within each of these. We recursively fill uncovered regions with smaller squares, guided by the disc's geometry. This computation is performed within one octant and symmetrically extended to cover the full disc. The resulting lattice square partition closely approximates the disc, while residual points outside the partition are handled separately. For detailed algorithms and illustrations, refer to Appendix B.2.

In our framework, this partitioning applies when Alice wants to test whether Bob lies within ℓ_2 distance δ of her location. This corresponds to checking for lattice points inside a disc centered at some point $\boldsymbol{w} \in \mathbb{Z}^2$ with radius δ.[4]

[4]While the method is presented for two dimensions, the underlying idea can be extended to higher dimensions with suitable generalizations.

Given a closed and compact surface S in $\mathbb{R}^d$, defined by a description S_{desc}, the algorithm produces a finite set $H^* = \{h_1, \ldots, h_K\}$ of pairwise disjoint axis-aligned hyperrectangles partitioning $S' \subseteq S$, along with the set of all L residual d-dimensional lattice points $\text{res}_i \in S - S'$. Thus, all lattice points of S are contained in $H^* \cup \{\text{res}_i\}_{i=1}^{L}$.

Phase 1 (Choosing the best resolution r^*)

- Compute the minimal and maximal coordinates of S along each dimension:

$$\boldsymbol{b}_{\min} = \left(\min_{\boldsymbol{x}\in S} x_1, \ldots, \min_{\boldsymbol{x}\in S} x_d\right) \quad \text{and} \quad \boldsymbol{b}_{\max} = \left(\max_{\boldsymbol{x}\in S} x_1, \ldots, \max_{\boldsymbol{x}\in S} x_d\right)$$

- Define the axis-aligned bounding box of S in dimension-range format as

$$\mathcal{B} = [\boldsymbol{b}_{\min}, \boldsymbol{b}_{\max}] = \left\{\boldsymbol{x} \in \mathbb{R}^d \mid b_{\min,i} \le x_j \le b_{\max,i}, \quad i = 1, \ldots, d\right\}.$$

- Identify the resolution r minimizing a cost function (e.g., by exhaustive evaluation over all possible resolutions). The cost function, denoted as $\text{Cost}(r)$, is defined in advance and depends explicitly on the set of fused hyperrectangles $H_r = \text{CutAndFuse}(S_{\text{desc}}, r)$ obtained by applying the CutAndFuse subroutine at resolution r.

> **Subroutine:** CutAndFuse(S_{desc}, r)
>
> **Input:** A surface description S_{desc} and a resolution $\text{r} \in \mathbb{N}$
> **Output:** A set $H_r = \{h_1, \ldots, h_{K_r}\}$ of axis-aligned disjoint hyperrectangles partitioning a subset of S
>
> – Partition the bounding box $\mathcal{B}$ of S into uniform axis-aligned grid cells of side length r, of the form
>
> $$g_{\boldsymbol{k}} = [\boldsymbol{b}_{\min} + \boldsymbol{k}r,\ \boldsymbol{b}_{\min} + (\boldsymbol{k}+\boldsymbol{1})r], \text{ where } \boldsymbol{k} \in \mathbb{Z}^d \text{ indexes the cells.}$$
>
> Let $\mathcal{G}_r \coloneqq \{g_{\boldsymbol{k}} \mid \boldsymbol{b}_{\min} + (\boldsymbol{k}+\boldsymbol{1})r \le \boldsymbol{b}_{\max}\}$ denote the set of all such grid cells.
> – Define the set of grid cells fully contained in S as $\mathcal{G}_r^{\text{in}} = \{g \in \mathcal{G}_r \mid g \subseteq S\}$.
> – Repeatedly fuse neighboring cells in $\mathcal{G}_r^{\text{in}}$ that are adjacent along one axis and aligned along all others, producing progressively larger hyperrectangles until no additional merges can be performed.
> – Return the resulting set of fused hyperrectangles: $H_r = \{h_1, \ldots, h_{K_r}\}$

- Let $r^* \coloneqq \arg\min_r \text{Cost}(r)$ be the optimal resolution.

Phase 2 (Compute hyperrectangles and residual points for resolution r^*)

- Let $H^* = \{h_1, \ldots, h_K\}$ denote the set of hyperrectangles output by $\text{CutAndFuse}(S_{\text{desc}}, r^*)$.
- Compute the residual lattice points in $S \setminus H^*$, as: $\{\text{res}_i\}_{i=1}^{L} = \{\boldsymbol{z} \in \mathbb{Z}^d \cap S \mid \boldsymbol{z} \notin \bigcup_{h\in H^*} h\}$.
- Output the set of disjoint hyperrectangles $H^* = \{h_1, \ldots, h_K\}$ and the set of residual points $\{\text{res}_i\}_{i=1}^{L}$.

Figure 3: The Partitioner($\cdot$) **procedure in our framework**

3.2.2 Partitioning a composite hyperrectangle region. We consider the problem of partitioning a composite hyperrectangle region, defined as the union of overlapping axis-aligned hyperrectangles in high-dimensional space. This setting arises naturally in practical applications, where a user's input region may correspond to a composite structure (e.g., building in 3D), making efficient partitioning essential for PPPT applications discussed in Section 5.

This problem can be tackled by a method proposed by Holub *et al.* [12] in the context of interpretable machine learning. The method uses a cut-and-fuse algorithm, similar to our earlier approach, but specifically tailored to align cuts with hyperrectangle edges, thus minimizing unnecessary subdivisions.

4 Geometric Fuzzy Matching

In this section, we introduce a general framework for geometric fuzzy matching. We begin with the distance-based matching setting of the ℓ_∞ distance, which provides the simplest scenario in our framework, as it does not require partitioning. Nevertheless, the construction is organized to parallel the structure of the general protocol, enabling a seamless extension to it later.

4.1 GFM for Infinity Distance

The key idea for improving the FM protocol of [29] for the ℓ_∞ distance is to reduce overall communication by minimizing the number of elements encoded in the OKVS. Specifically, we avoid encoding every value in the interval $[w_i - \delta, w_i + \delta]$ by representing the interval with the minimal set of $O(\log \delta)$ binary prefixes instead, for each dimension $i \in [d]$. This prefix-based representation yields a more compact OKVS, which reduces both communication and computational effort on the receiver's side. However, the trade-off is that the sender must now evaluate $O(\log \delta)$ prefixes per dimension, resulting in a modest but acceptable increase in communication.

Turning to the protocol's structure, although it builds directly on the FM protocol of [29], we introduce key adaptations to enable (i) the integration of the Distance-Aware Set Augmentation (DASA) technique into the protocol, and (ii) compatibility with our more general framework, which extends to ℓ_p distances and beyond.

To achieve the second goal, we describe the receiver's input set $\mathcal{R}_{\text{input}}$ by a concise description $\mathcal{R}_{\text{desc}}$—for ℓ_∞, this includes $\{\boldsymbol{w}, \{\delta, \text{“}\ell_\infty\text{”}\}\}$, a private part $\mathcal{R}_{\text{priv}}$ representing the receiver's exact location, and a structural part $\mathcal{R}_{\text{struct}}$. The receiver sends the structure metadata R_{info} to the sender in the first message. Importantly, $\mathcal{R}_{\text{struct}}$ alone does not reveal the receiver's location.

To achieve the first goal, we first introduce a function RecStrings_∞ that takes $\mathcal{R}_{\text{desc}}$ as input and outputs both the representative strings for $\mathcal{R}$'s set and the associated structural information R_{info}. We also introduce a corresponding function for the sender's side, SndStrings_∞, which takes as input the sender's point $\boldsymbol{q}$ along with R_{info}, and outputs the sender's augmented set of representative strings S.

The functions RecStrings_∞ and SndStrings_∞ are described below and the GFM_∞ protocol is illustrated in Figure 4.

Algorithm 2 $\text{RecStrings}_\infty(\mathcal{R}_{\text{desc}})$

1: **Input:** $\mathcal{R}_{\text{desc}} \coloneqq \{\mathcal{R}_{\text{priv}}, \mathcal{R}_{\text{struct}}\} = \{\boldsymbol{w}, \{\delta, \text{“}\ell_\infty\text{”}\}\}$, which describes the receiver's input set, with $\boldsymbol{w} \in \mathbb{Z}^d$ and $\delta \in \mathbb{Z}_+$
2: **Output:** A tuple (R, R_{info}) of the receiver's augmented set of strings R and information regarding the structure of $\mathcal{R}$'s set
3: **for** $i = 1, \ldots, d$ **do**
4: $\quad \{\text{str}_{i,j}\}_{j=1}^{J_i} \leftarrow \text{DASA}_{\mathcal{R}}(w_i, \delta)$
5: **end for**
6: $R_i \coloneqq \{i \| \text{str}_{i,j}\}_{j=1}^{J_i}$
7: Assign $\boldsymbol{R} \coloneqq (R_1, \ldots, R_d)$ and $R_{\text{info}} \coloneqq \{\delta, \text{“}\ell_\infty\text{”}\}$
8: Set $R \coloneqq \boldsymbol{R}$
9: **return** $(R, R_{\text{info}}, \boldsymbol{\delta} = (\delta, \ldots, \delta))$

Algorithm 3 $\text{SndStrings}_\infty(\boldsymbol{q}, R_{\text{info}} = \{\delta, \text{“}\ell_\infty\text{”}\})$

1: **Input:** Sender's vector $\boldsymbol{q} \in \mathbb{Z}^d$ and information R_{info} regarding the structure of $\mathcal{R}$'s set
2: **Output:** Sender's augmented set of strings S
3: Parse $R_{\text{info}} = \{\delta, \text{“}\ell_\infty\text{”}\}$
4: **for** $i = 1, \ldots, d$ **do**
5: $\quad \{\text{str}_{i,j}\}_{j=1}^{J_i} \leftarrow \text{DASA}_{\mathcal{S}}(q_i, \delta)$
6: **end for**
7: $\boldsymbol{s}_j \coloneqq \left(1 \| \text{str}_{1,j_1}, \ldots, d \| \text{str}_{d,j_d}\right), \forall j_1 \in [J_1], \ldots, j_d \in [J_d]$
8: Assign $S \coloneqq \{\boldsymbol{s}_j = (s_{j,1}, \ldots, s_{j,d})\}_{j=1}^{J_S}$ where $J_S = \prod_{i=1}^{d} J_i$
9: **return** S

Prefix-based Representation. For the GFM_∞ setting, the sender $\mathcal{S}$ and receiver $\mathcal{R}$ each hold one point in a d-dimensional space, namely $\boldsymbol{w}, \boldsymbol{q} \in \mathbb{Z}^d$, respectively. The goal of the protocol is for the receiver to determine whether $\|\boldsymbol{w} - \boldsymbol{q}\|_\infty \leq \delta$, for some positive integer threshold $\delta \in \mathbb{Z}_+$. Most prior work in this setting [7, 8, 29] utilize—either implicitly or explicitly—the fact that

$$\|(\boldsymbol{w}, \boldsymbol{q})\|_\infty \leq \delta \Leftrightarrow w_i - \delta \leq q_i \leq w_i + \delta, \forall i \in [d],$$

which allows the d-dimensional problem to be decomposed into d separate 1D instances. In the work of van Baarsen *et al.* [29], this is done implicitly by transforming the problem so that $\mathcal{R}$ holds d intervals $[w_i - \delta, w_i + \delta]$ and the parties run a membership test for each dimension simultaneously.

For our protocol we take this a step further by succinctly representing this interval by a collection of bit strings corresponding to their binary representations (i.e., utilizing the Distance-Aware Set Augmentation algorithms of [3]). Getting these succinct representative strings from the parties' original sets is done by the functions RecStrings_∞ and SndStrings_∞ for receiver and sender respectively.

Specifically, the receiver's function produces, as part of its output, a set of representative strings for each interval $[w_i - \delta, w_i + \delta]$. For each dimension $i \in [d]$, approximately $O(\log \delta)$ strings are created using $\text{DASA}_{\mathcal{R}}$ and we denote the exact number of them by J_i. Aside from the set of strings, the function also produces auxiliary information that encodes the structure of $\mathcal{R}$'s input. This includes the threshold δ and a flag indicating that the current instance uses the ℓ_∞ distance.[5] This additional information will be sent to $\mathcal{S}$ along with the first message.

On the other hand, the sender's function utilizes $\text{DASA}_{\mathcal{S}}$, to first obtain representative strings for each $q_i, \forall i \in [d]$ by replacing the required number of least significant bits in the binary representation of q_i with wildcard bits. However, unlike $\mathcal{R}$, the sender must construct the full set of vectors formed by taking the Cartesian product of the representative strings from each dimension—effectively merging the d dimensions. Thus, the final output will be a set S containing vectors $\boldsymbol{s}_j$, where each $\boldsymbol{s}_j$ is a tuple of representative strings, one from each dimension. We denote the total number of such vectors by $J_{\mathcal{S}}$, which equals the product of the number of representative strings across all dimensions. In this case, this number is approximately $O(\log \delta)^d$.

The Protocol. Both receiver and sender run the RecStrings and SndStrings functions at the start of their respective processes, getting a vector $\boldsymbol{R} = (R_1, \ldots, R_d)$ and a set S respectively. The remainder of the protocol mirrors that of [29], but the receiver now encodes $R_i \coloneqq \{i \| \text{str}_{i,j}\}_j$ in each dimension, rather than the interval $[w_i - \delta, w_i + \delta]$, and the sender works with $J_{\mathcal{S}}$ vectors $\boldsymbol{s}_j$ instead of a single $\boldsymbol{q}$.

Theorem 2 (GFM_∞ Correctness). Suppose that the OKVS used in the protocol of Figure 4 is correct and satisfies the independence property from Lemma 2. Let $H_\gamma : \{0,1\}^* \to \{0,1\}^\gamma$ and $H_{\kappa'} : \mathbb{G} \to \{0,1\}^{\kappa'}$ be universal hash functions, where $\gamma = \kappa + (d+1)\log\log\delta$ and $\kappa' = \kappa + d\log\log\delta$ with κ being the statistical parameter. Then, the protocol of Figure 4 is correct with probability $1 - \text{negl}(\kappa)$.

Proof. The correctness proof resembles those of Theorem 5 and 11 from [29], with the key distinction lying in the representation of the receiver's interval $[w_i - \delta, w_i + \delta]$ in each dimension—similarly for the sender. Whereas [29] encodes this interval explicitly as the values $\{w_i - \delta, \ldots, w_i + \delta\}$, our protocol encodes it via the set of strings generated by DASA. Consequently, to ensure correctness, the sender cannot use raw input coordinates directly in the OKVS, but must instead construct vectors over the Cartesian product of the representative strings output by $\text{DASA}_{\mathcal{S}}$. From there, the proof follows directly from the correctness of DASA and the arguments in Theorem 5 of [29]. The complete proof is given in Appendix C.1. □

Theorem 3 (GFM_∞ Security). Suppose that the OKVS is oblivious (see definition 1) and the DDH assumption holds. Then, the protocol of Figure 4 realizes the $\mathcal{F}_{\text{GFM}}$ functionality defined in Section 2.4 for the ℓ_∞ distance function against semi-honest adversaries.

[5] This enables the protocol to adapt dynamically based on the receiver's input.

$\mathbf{Receiver}_1(\mathcal{R}_{\text{desc}})$

Get $\boldsymbol{R}, R_{\text{info}}, \boldsymbol{\delta} \leftarrow \text{RecStrings}(\mathcal{R}_{\text{desc}})$
Sample $g \leftarrow_{\$} \mathbb{G}$ and $r \leftarrow_{\$} \mathbb{Z}_q$
Compute $h = g^r$
Get $\{\text{list}_i\}_{i=1}^{d} \leftarrow \text{GetList}(r, \boldsymbol{\delta}, \boldsymbol{R}, 0^\kappa)$
Get $E_i \leftarrow \text{Encode}(\text{list}_i)$, for each $i \in [d]$
Set $\boldsymbol{E} = \{E_1, \ldots, E_d\}$
Output $\text{msg}_1 \coloneqq (R_{\text{info}}, g, h, \boldsymbol{E})$ and secret state $\text{st} \coloneqq r$

$\mathbf{Receiver}_2(\text{st}, \text{msg}_2)$

Parse $\text{msg}_2 \coloneqq \left\{(u_j, v_j)\right\}_{j=1}^{J_S}$ and $\text{st} \coloneqq r$
Output 1 if $\exists j$ s.t. $H_{\kappa'}(u_j^r) = v_j$; otherwise 0

$\mathbf{Sender}_1(\boldsymbol{q} \in \mathbb{Z}^d, \text{msg}_1)$

Parse $\text{msg}_1 = (R_{\text{info}}, g, h, \boldsymbol{E})$
Compute $S \leftarrow \text{SndStrings}(\boldsymbol{q}, R_{\text{info}})$
Parse $S \coloneqq \{\boldsymbol{s}_j = (s_{j,1}, \ldots, s_{j,d})\}_{j=1}^{J_S}$.
For $j = 1, \ldots, J_S$:
  Get $(u_j, v_j) \leftarrow \text{GetTuples}(g, h, \boldsymbol{E}, \boldsymbol{s}_j, 0^\kappa)$
Set $\text{msg}_2 \coloneqq \left\{(u_j, v_j)\right\}_{j=1}^{J_S}$
Shuffle msg_2 by j
Output msg_2

Algorithm 4 $\text{GetList}(r, \boldsymbol{\delta}, \boldsymbol{R}, \text{label})$

Input: A secret $r \in \mathbb{Z}_q$, a vector of thresholds $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_d)$, a vector of sets of strings $\boldsymbol{R}$ and an identifier label
Output: d lists of tuples, denoted list_i for $i \in [d]$

Parse $\boldsymbol{R} \coloneqq (R_1, \ldots, R_d)$ with $R_i = \{i \| \text{str}_{i,j}\}_{j=1}^{J_i}$
for $i = 1, \ldots, d$ **do**
  for $j = 1, \ldots, J_i$ **do**
    $\text{key}_{i,j} \leftarrow H_\gamma(\text{label} \| i \| \text{str}_{i,j})$
    $h_j \leftarrow_{\$} \mathbb{G}$
    $\text{val}_{i,j} = \left(h_j \,||\, h_j^r\right)$
  end for
  Set $\text{list}_i = \{(\text{key}_{i,j}, \text{val}_{i,j})\}_{j=1}^{J_i}$
  Pad list_i' up to $2\lfloor \log(2\delta_i + 1) \rfloor$ with random pairs
end for
return $\{\text{list}_i\}_{i=1}^{d}$

Algorithm 5 $\text{GetTuples}(g, h, \boldsymbol{E}, \boldsymbol{s}_j, \text{label}')$

Input: A vector of OKVS encodings $\boldsymbol{E} = (E_1, \ldots, E_d)$, a vector of strings $\boldsymbol{s}_j = (s_{j,1}, \ldots, s_{j,d})$ and $h \in \mathbb{G}$
Output: A tuple (u_j, v_j)

for $i = 1, \ldots, d$ **do**
  $\text{key}_{j,i} \leftarrow H_\gamma(\text{label}' \| s_{j,i})$
  $(u_{j,i} || v_{j,i}) \leftarrow \text{Decode}(E_i, \text{key}_{j,i})$
end for
Sample $a_j, b_j \leftarrow_{\$} \mathbb{Z}_q$
$u_j = g^{a_j} \left(\prod_{i=1}^{d} u_{j,i}\right)^{b_j}$
$v_j' = h^{a_j} \left(\prod_{i=1}^{d} v_{j,i}\right)^{b_j}$
$v_j \leftarrow H_{\kappa'}(v_j')$
return (u_j, v_j)

Figure 4: Geometric Fuzzy Matching for infinity distance (GFM_∞)

Proof. Intuitively, the change in encoding of the intervals should not impact the security from that of [29], but only the correctness, if the appropriate padding is added. Additionally, the transmission of R_{info} (instead of having this information as a parameter) is accounted for by $\mathcal{F}_{\text{INFO}}$. For the full proof, see Appendix C.2. □

Theorem 4 (GFM_∞ Complexity). Suppose λ and κ are the security and statistical parameters, respectively, for the protocol of Fig. 4. Then, $O\left((2d \log \delta)\lambda + (\log \delta)^d(\lambda + \kappa')\right)$ will be the communication complexity, for $\kappa' = \kappa + d \log\log \delta$. Additionally, the computational complexity is $O(2d \log \delta + (\log \delta)^d)$ for the receiver and $O(d(\log \delta)^d)$ for the sender.

Proof. First, we note that DASA is not taken into account in the computational complexity since it is considered as part of preprocessing in our applications. Moreover, the complexity bounds hold assuming that OKVS supports constant rate, linear-time encoding, and constant-time decoding. Also, without prior knowledge of δ, the sender can still precompute keys up to a guessed or maximal value $\text{MaxBitLen} - 1$ to reduce online computation. □

4.2 GFM for General d-Dimensional Regions

The previous protocol does not extend directly to Minkowski distances, since by definition,

$$\|\boldsymbol{w} - \boldsymbol{q}\|_p \coloneqq \left(\sum_{i=1}^{d} |w_i - q_i|^p\right)^{1/p},$$

and this does not decompose into d separate one-dimensional components. To circumvent this limitation, van Baarsen *et al.* [29] leveraged the homomorphic properties of DDH tuples, enabling the sender to homomorphically compute the p-powered ℓ_p distance, $\left(\sum_{i=1}^{d} |w_i - q_i|^p\right)$. This allowed the receiver to verify if the result is less than δ^p by checking if it appears in a list of encrypted values $0, \ldots, \delta^p$. However, creating and transmitting this list increased communication and computation by an additive term $O(\delta^p)$.

Our approach avoids this overhead by reinterpreting fuzzy matching not as a distance problem, but as a geometric inclusion problem: "is the sender's point inside the region defined by the receiver?" For instance, in the case of ℓ_2-fuzzy matching, the acceptance region is a hyperball. The problem then reduces to testing whether the sender's point lies among the integer lattice points inside this ball. This mindset enables us, not only to achieve more efficient ℓ_p-fuzzy

matching to generalize fuzzy matching to arbitrary d-dimensional geometric regions in $\mathbb{Z}^d$, a direction that, to our knowledge, has not been previously explored.

To handle such general regions, we introduce a space-partitioning strategy. Suppose, for simplicity, that the receiver's region is a circle. We approximate this region using a collection of axis-aligned lattice rectangles, each of which can be handled using a standard instance of fuzzy matching under the ℓ_∞ metric. Then, we run parallel instances of the GFM_∞ protocol—one per rectangle. However, since these rectangles typically do not cover the entire region precisely, we identify and separately handle the residual points that remain uncovered. These residual points are grouped together and handled using an additional fuzzy matching instance. Each instance—whether corresponding to a rectangle or the residual set—is tagged with a unique label, since grouping multiple instances together arbitrarily would compromise correctness.

In our framework, this approximation is performed by the adaptive procedure Partitioner of Section 3. It takes as input a structured description of the region and outputs two sets: a list of axis-aligned hyperrectangles, each parameterized by a center $\boldsymbol{c}_k$ and a side-length vector $\boldsymbol{\delta}_k$ for $k \in [K]$, and a set of residual points $\{\mathsf{res}_\ell\}_{\ell=1}^{L}$. The union of the lattice points in these rectangles and the residuals covers the receiver's full input $\mathcal{R}_{\mathsf{input}}$.

The overall complexity of the protocol depends on the number of rectangles K and residual points L, which are determined by the structure of the region and the efficiency of the partitioning algorithm. In what follows, we describe how the $\mathsf{RecStrings}_\infty$ and $\mathsf{SndStrings}_\infty$ algorithms are adapted to handle partition-based input encoding and multiple labeled fuzzy matching instances. The full protocol is in Figure 6.

Prefix-based Representation. In this generalized setting, after utilizing Partitioner, the receiver's region is described using hyperrectangles and a set of residual integer points. For each hyperrectangle, the region is specified by a center $\boldsymbol{c}_k$ and a vector of half-side lengths $\boldsymbol{\delta}_k$. The receiver applies the encoding procedure $\mathsf{DASA}_{\mathcal{R}}$ on each coordinate of every hyperrectangle to produce prefix strings that represent all points inside the corresponding box. Each such encoded set becomes one block of receiver strings, augmented with coordinate indices. In contrast, residual lattice points (those not captured by any hyperrectangle) are included in the receiver's augmented string set without modification and are handled in a slightly different manner later.

The sender, on the other hand, receives only a structural summary R_{info} of the receiver's region, which consists of the list of side lengths $\{\boldsymbol{\delta}_k\}_{k=1}^{K}$ and the underlying metric. Then, to ensure correctness across all receiver blocks, the sender uses the maximal side length per coordinate across all blocks, $\delta_{\max,i}$, to compute a conservative overapproximation via $\mathsf{DASA}_{\mathcal{S}}$.[6] The final sender set is then the Cartesian product of all coordinate-wise encodings. Also, we note that residual points will not be captured by these vectors, and will be processed differently.

On Residual Points. Unlike points that fall within a hyperrectangle and are processed using the standard GetList and GetTuple routines, residual points are handled using simplified variants: $\mathsf{GetList}_{\mathsf{res}}$ and $\mathsf{GetTuples}_{\mathsf{res}}$. These algorithms omit any dependence on geometric structure and treat each residual point independently, relying solely on string-level identifiers. In particular, $\mathsf{GetList}_{\mathsf{res}}$ applies the same cryptographic encoding as in the main routine but over residual strings tagged with a common label, while $\mathsf{GetTuples}_{\mathsf{res}}$ decodes and re-randomizes them analogously.

The Protocol. Our GFM protocol realizes $\mathcal{F}_{\mathsf{GFM}}$ by expressing the receiver's region as a union of structured hyperrectangles and residual points through the Partitioner algorithm. Each hyperrectangle is encoded using $\mathsf{DASA}_{\mathcal{R}}$ to generate per-coordinate prefix strings, while residual points are represented directly as strings.

The sender, upon receiving structural metadata and the encodings, expands their input vector into a set of augmented strings using $\mathsf{DASA}_{\mathcal{S}}$, forming a Cartesian product across coordinates. For each block—both hyperrectangles and residuals—the sender evaluates blind match tuples using GetTuples and $\mathsf{GetTuples}_{\mathsf{res}}$, returning them in a randomized order. The receiver then performs a DDH self-reduction check to determine if the sender's vector lies within the encoded region.

[6] Although the sender's encoding covers a superset of each block, matches are determined solely by the receiver's $\mathsf{DASA}_{\mathcal{R}}$ encodings. Therefore, no false positives are introduced, preserving correctness.

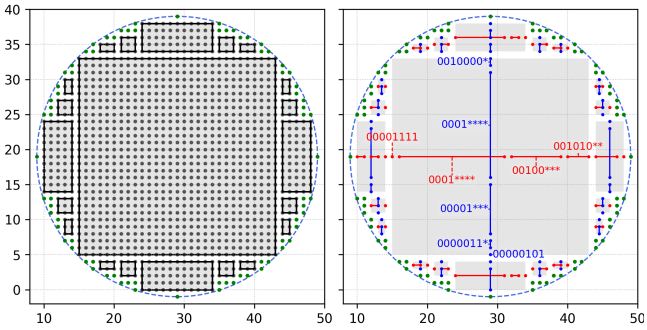


Figure 5: Illustration of RecStrings **steps on input** $\mathcal{R}_{\mathsf{desc}} \coloneqq \{(29, 19), 20, \text{“}\ell_2\text{”}\}$**, using the partitioner algorithm of Section 3.2.1. The residual points are denoted with green.**

Algorithm 6 $\mathsf{RecStrings}(\mathcal{R}_{\mathsf{desc}} \coloneqq (\mathcal{R}_{\mathsf{priv}}, \mathcal{R}_{\mathsf{struct}})$

Input: A description $\mathcal{R}_{\mathsf{desc}} \coloneqq (\mathcal{R}_{\mathsf{priv}}, \mathcal{R}_{\mathsf{struct}})$ of receiver's set
Output: A tuple (R, R_{info}) of the receiver's augmented set of strings R and information regarding the structure of $\mathcal{R}$'s set

$(\{(\boldsymbol{c}_k, \boldsymbol{\delta}_k)\}_{k=1}^{K}, \{\mathsf{res}_\ell\}_{\ell=1}^{L}) \leftarrow \mathsf{Partitioner}(\mathcal{R}_{\mathsf{desc}})$
for $(\boldsymbol{c}_k, \boldsymbol{\delta}_k)$ **do**: *# ∀ hyperrectangle of center $\boldsymbol{c}_k$ & half-side $\boldsymbol{\delta}_k$*
  for $i = 1, \ldots, d$ **do**
    $\{\mathsf{str}_{k,i,j}\}_{j=1}^{J_{k,i}} \leftarrow \mathsf{DASA}_{\mathcal{R}}(c_{k,i}, \delta_{k,i})$
  end for
end for
$\boldsymbol{R}_k \coloneqq (R_{k,1}, \ldots, R_{k,d}), \forall k \in [K]$ & $R_{k,i} = \{i \| \mathsf{str}_{k,i,j}\}_{j=1}^{J_{k,i}}$
Assign $R = \{\boldsymbol{R}_1, \ldots, \boldsymbol{R}_K, \{\text{“}\mathsf{res}_\ell\text{”}\}_{\ell=1}^{L}\}$
for $i = 1, \ldots, d$ **do** *# Computing $\delta_{\max}$ for the sender*
  Set $\delta'_i = \max_k\{\delta_{k,i}\}$
end for
Create unique identifiers $\{\mathsf{label}_k\}_{k=1}^{K}, \mathsf{label}_{\mathsf{res}}$.
$R_{\mathsf{info}} \coloneqq (\boldsymbol{\delta}_{\max} \coloneqq (\delta'_1, \ldots, \delta'_d), \mathcal{R}_{\mathsf{struct}}, \{\mathsf{label}_k\}_{k=1}^{K}, \mathsf{label}_{\mathsf{res}})$

return $(R, R_{\mathsf{info}}, \{\boldsymbol{\delta}_k\}_{k=1}^{K})$

$\mathbf{Receiver}_1(\mathcal{R}_{\text{desc}})$

Get $R, R_{\text{info}}, \{\boldsymbol{\delta}_k\}_{k=1}^{K} \leftarrow \text{RecStrings}(\mathcal{R}_{\text{desc}})$
Parse $R \coloneqq \{\boldsymbol{R}_1, \ldots, \boldsymbol{R}_K, \{\text{“res}_\ell\text{”}\}_{\ell=1}^{L}\}$
Parse $R_{\text{info}} \coloneqq (\boldsymbol{\delta}_{\max}, \mathcal{R}_{\text{struct}}, \{\text{label}_k\}_{k=1}^{K}, \text{label}_{\text{res}})$
Sample $g \leftarrow\!\!\$\ \mathbb{G}$ and $r \leftarrow\!\!\$\ \mathbb{Z}_q$
Compute $h = g^r$
Set $\text{list}_{\text{res}} = \emptyset$ and $\text{list}_i = \emptyset$ for each $i \in [d]$
For each $k = 1, \ldots, K$:
  Get $\{\text{list}'_i\}_{i=1}^{d} \leftarrow \text{GetList}(r, \boldsymbol{\delta}_k, \boldsymbol{R}_k, \text{label}_k)$
  Set $\text{list}_i = \text{list}_i \cup \text{list}'_i$ for each $i \in [d]$
Get $E_i \leftarrow \text{Encode}(\text{list}_i)$, for each $i \in [d]$
Set $\boldsymbol{E} = \{E_1, \ldots, E_d\}$.
Get $\text{list}_{\text{res}} \leftarrow \text{GetList}_{\text{res}}(r, \{\text{“res}_\ell\text{”}\}_{\ell=1}^{L}, \text{label}_{\text{res}})$
Set $E_{\text{res}} \leftarrow \text{Encode}(\text{list}_{\text{res}})$
Set $\text{msg}_1 \coloneqq (g, h, R_{\text{info}}, \boldsymbol{E}, E_{\text{res}})$
Output msg_1 and secret state $\text{st} \coloneqq r$

$\mathbf{Sender}_1(\boldsymbol{q} \in \mathbb{Z}^d, \text{msg}_1)$

Parse $\text{msg}_1 \coloneqq \left(g, h, \boldsymbol{E}, R_{\text{info}}, \{\text{label}_k\}_{k=1}^{K+1}\right)$
Compute $S \leftarrow \text{SndStrings}(\boldsymbol{q}, R_{\text{info}})$
Parse $S \coloneqq \{\boldsymbol{s}_j = (s_{j1}, \ldots, s_{jd})\}_{j=1}^{J_S}$.
For each $k = 1, \ldots, K$: *# check in each hyperrectangle*
  For each $j = 1, \ldots, J_S$:
    Get $(u_{k,j}, v_{k,j}) \leftarrow \text{GetTuples}(g, h, \boldsymbol{E}, \boldsymbol{s}_j, \text{label}_k)$
Set $\text{msg}_2 \coloneqq \left\{(u_{k,j}, v_{k,j})_{k\in[K], j\in[J_S]}\right\}$
Shuffle msg_2 by j and then by k
Get $(u_{K+1}, v_{K+1}) \leftarrow \text{GetTuples}_{\text{res}}(g, h, E_1, \text{“}\boldsymbol{q}\text{”}, \text{label}_{K+1})$
Insert (u_{K+1}, v_{K+1}) at a random position in msg_2
Output msg_2

$\mathbf{Receiver}_2(\text{st}, \text{msg}_2)$

Parse $\text{msg}_2 \coloneqq \{(u_t, v_t)\}_{t=1}^{K\cdot J_S+1}$, and $\text{st} = r$
Output 1 if $\exists\, t \in [K \cdot J_S + 1]$ s.t. $H_{\kappa'}(u_t^r) = v_t$; otherwise 0

Figure 6: Our protocol for Geometric Fuzzy Matching

Algorithm 7 $\text{SndStrings}(\boldsymbol{q}, R_{\text{info}} = (\{\boldsymbol{\delta}_{\max}, \mathcal{R}_{\text{struct}}\}))$

Input: Sender's vector $\boldsymbol{q} \in \mathbb{Z}^d$ and information R_{info} regarding the structure of $\mathcal{R}$'s set and the related partitioning
Output: Sender's augmented set of strings S

Parse $R_{\text{info}} = (\boldsymbol{\delta}_{\max}, \mathcal{R}_{\text{struct}})$
for $i = 1, \ldots, d$ **do**
  $\{\text{str}_{i,j}\}_{j=1}^{J_i} \leftarrow \text{DASA}_{\mathcal{S}}(q_i, \boldsymbol{\delta}_{\max,i})$
end for
$\boldsymbol{s}_j \coloneqq \left(1 \| \text{str}_{1,j_1}, \ldots, d \| \text{str}_{d,j_d}\right), \forall j_1 \in [J_1], \ldots, j_d \in [J_d]$
Assign $S \coloneqq \{\boldsymbol{s}_j = (s_{j,1}, \ldots, s_{j,d})\}_{j=1}^{J_S}$, where $J_S = \prod_{i=1}^{d} J_i$
return S

Algorithm 8 $\text{GetList}_{\text{res}}(r, \{\text{“res}_\ell\text{”}\}_{\ell=1}^{L}, \text{label})$

Input: A secret $r \in \mathbb{Z}_q$ and a set of strings $\{\text{“res}_\ell\text{”}\}_{\ell=1}^{L}$
Output: A list of tuples, denoted list_{res}

for $\ell = 1, \ldots, L$ **do**
  $\text{key}_\ell \leftarrow H_\gamma(\text{label} \| \text{“res}_\ell\text{”})$
  $h_\ell \leftarrow\!\!\$\ \mathbb{G}$
  $\text{val}_\ell = \left(h_\ell \,||\, h_\ell^r\right)$
end for
return $\text{list}_{\text{res}} \coloneqq \{(\text{key}_\ell, \text{val}_\ell)\}_{\ell=1}^{L}$

Algorithm 9 $\text{GetTuples}_{\text{res}}(g, h, E, \text{“}\boldsymbol{q}\text{”}, \text{label}')$

Input: An OKVS encoding E, a string “$\boldsymbol{q}$” and $h \in \mathbb{G}$
Output: A tuple $(u_{\text{res}}, v_{\text{res}})$

$\text{key} \leftarrow H_\gamma(\text{label}' \| \text{“}\boldsymbol{q}\text{”})$
$(u || v) \leftarrow \text{Decode}(E, \text{key})$
Sample $a, b \leftarrow\!\!\$\ \mathbb{Z}_q$
$u_{\text{res}} = g^a u^b$
$v' = h^a v^b$
$v_{\text{res}} \leftarrow H_{\kappa'}(v')$
return $(u_{\text{res}}, v_{\text{res}})$

Theorem 5 (GFM Correctness). Suppose that a partitioning algorithm has been used that creates a set H^* of K disjoint hyperrectangles with $\{\boldsymbol{\delta}_k\}_{k=1}^{K}$, and L residual points. Set $\delta_{\max} \coloneqq \max_i \delta_{\max,i}$. Additionally, suppose that the OKVS is correct and satisfies the independence property from Lemma 2, and that $H_\gamma : \{0,1\}^* \to \{0,1\}^\gamma$ and $H_{\kappa'} : \mathbb{G} \to \{0,1\}^{\kappa'}$ are universal hash functions, where $\gamma = \kappa + \max\{\log L, \log K + (d+1)\log\log \delta_{\max}\}$ and $\kappa' = \kappa + \log K + d \log\log \delta_{\max}$ with κ being the statistical parameter. Then, the protocol of Figure 6 is correct with probability $1 - \text{negl}(\kappa)$.

Proof. It is easy to see that, if $\boldsymbol{q} \in \mathcal{R}_{\text{input}}$, then $\boldsymbol{q}$ will either be an integer point lying inside one of the disjoint hyperrectangles of H^* or one of the residual points $\{\text{res}_\ell\}_{\ell=1}^{L}$. In the first case, one can prove that correctness holds by using similar arguments with those made in Appendix C.1. In the second case, correctness again reduces to a simpler variant of Theorem 2, since $\text{GetList}_{\text{res}}$ and $\text{GetTuple}_{\text{res}}$ are just simpler variants of GetList and GetTuple respectively.

If on the other hand, $\boldsymbol{q} \notin \mathcal{R}_{\text{input}}$, then again correctness holds with overwhelming probability by using universal hash functions H_γ and $H_{\kappa'}$, following the proof in Appendix C.1.

The complete proof is given in Appendix D.1. □

Theorem 6 (GFM Security). Suppose that the OKVS is oblivious (see definition 1) and the DDH assumption holds. Then, the protocol of Figure 6 realizes the functionality defined in Section 2.4 against semi-honest adversaries.

Theorem 7 (GFM Complexity). Suppose that λ and κ are the security and statistical parameters, respectively, for the protocol of Figure 6. Additionally, suppose that a partitioning algorithm has been used that creates a set H^* of K disjoint hyperrectangles with $\{\boldsymbol{\delta}_k\}_{k=1}^{K}$, and L residual points. Set $\delta_{\max} \coloneqq \max_i \delta_{\max,i}$. Then, $O\left((dK \log \delta_{\max})\lambda + dK(\log \delta_{\max})^d(\lambda + \kappa')\right)$ will be the communication complexity, for $\kappa' = \kappa + \log K + d \log\log \delta_{\max}$. Additionally,

the computational complexity is $O(2dK \log \delta_{\max} + K(\log \delta_{\max})^d + L)$ for the receiver and $O(dK(\log \delta_{\max})^d)$ for the sender.

Proofs of the theorems are deferred to Appendices D.2 and D.3.

Remark 2. We remark the following:

(i) In certain applications, it may be advantageous to relax correctness by omitting residual points and prioritizing the maximization of the aggregate volume covered by the hyperrectangles.
(ii) The framework can be extended to Fuzzy PSI where the receiver has multiple disjoint regions and the sender has multiple points, a kind of Geometric Fuzzy PSI. However, handling arbitrary surfaces in this setting currently incurs significant overhead, so we leave this for future work.

5 Privacy-preserving Proximity Testing from Geometric Fuzzy Matching

In this section, we demonstrate how GFM can be effectively applied within the context of Privacy-Preserving Proximity Testing (PPPT).

Prior Approaches to PPPT. Early approaches to proximity tracking were mostly centralized and ignored privacy, with few exceptions such as Hide&Crypt by Freni *et al.* [5], which offered limited protection as the server still knew users' cloaked regions.

The notion of privacy-preserving proximity testing (PPPT) was first formally introduced by Narayanan *et al.* [22], enabling one party to determine whether they are within a specified distance of another without revealing additional location information. Subsequent works improved on this foundation: Saldamli *et al.* [27] optimized performance using geometric and linear algebra techniques, while Šedênka *et al.* [30] introduced homomorphic distance computations accounting for Earth's curvature.

A key limitation of these approaches is that both parties must be online simultaneously during protocol execution. Oleynikov *et al.* [23–25] addressed this with asynchronous two-server PPPT protocols. However, while this removes the online requirement, it relies on the strong assumption that the servers do not collude. Another limitation of these and most other PPPT works is that they focus solely on distance-based proximity testing. An exception is VicinityLocator by Šikšnys *et al.* [31], which allows proximity to be represented in terms of arbitrary shapes; however, it relies on a shared encryption function, introducing potential privacy leaks.

To overcome these constraints, our geometric fuzzy matching protocols offer a solution by removing the requirement for non-colluding servers, while enabling support for a wider variety of proximity models. We demonstrate this with two examples: distance-based and region-based proximity testing.

5.1 Distance-based Private Proximity Testing

Our protocol can be directly applied in settings where Alice wishes to determine whether Bob is within a specified proximity, defined with respect to a chosen distance metric, without revealing additional information about their respective positions. While our primary focus is on the two-party case, the proposed approach naturally extends to a multiparty scenario, allowing Alice to test proximity with multiple individuals (e.g., friends) by executing the protocol independently with each of them. This setting aligns with the standard model adopted in prior work on PPPT. However, our framework offers greater generality by supporting a broader range of distance metrics beyond the commonly used Euclidean distance in two dimensions (i.e., $\ell_2, d = 2$). In particular, our approach accommodates three-dimensional spaces and alternative distance functions, which are especially relevant also in more niche applications such as drone coordination and aerial navigation. However, it is worth noting that to fully support dynamic environments involving moving entities, the framework would require further extension. Specifically, the proximity region would need to account for motion-induced uncertainty, ensuring robustness in scenarios where positions may vary over time.

5.2 Region-based Private Proximity Testing

Our Setting. We adapt our framework to the PPPT setting where proximity is defined in terms of region inclusion rather than distance. Specifically, the receiver (Alice) wishes to determine whether the sender (Bob) resides within a specified, potentially complex region, such as a campus, mall, or park. The protocol must ensure that no further information about the parties' locations is revealed beyond this binary inclusion test.

Example I. Alice (receiver) specifies a 2D layout (e.g., a park or building floor plan) of the area she is currently in, with the goal of privately testing whether Bob (sender) is located inside. Since disclosure of this region would allow Bob to infer Alice's precise location, her input must remain concealed. To achieve this, our GFM protocol leverages the generic Partitioner to approximate the region via disjoint rectangles and residual points, which are then encoded into fuzzy match strings. This again enables a privacy-preserving inclusion check that reveals nothing beyond the outcome.

Example II. In domains that naturally admit a hyperrectangular decomposition (e.g. multi-floor buildings or 3D architectural models) Alice can define her input region as a union of overlapping hyperrectangles (see Fig. 7). In this case, the optimized partitioning of Section 3.2 can be used, efficiently decomposing this union into disjoint hyperrectangles without needing residual points. These are then directly encoded into strings within the GFM protocol, enabling an efficient and privacy-preserving inclusion test.

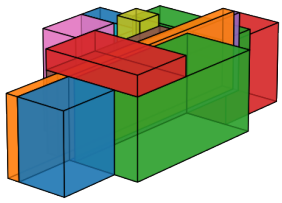


Figure 7: Disjoint hyperrectangles approximating a shopping mall, derived using the algorithm of Section 3.2.2.

Limitations of GFM. We remark two limitations, which also affect the range of applications to which GFM can be deployed, and discuss possible approaches to address them in Appendix E.

First, in the functionality considered in this work, the sender learns the full structure of the region through R_{info}. This may be undesirable in applications that require hiding the underlying layout completely. However, in practice, the sender only requires the

Table 1: Communication overhead (MB) and runtime (ms) of (geometric) fuzzy matching for ℓ_2, ℓ_3, ℓ_4, ℓ_5, and ℓ_{10} distances.

	ℓ_2 Distance		ℓ_3 Distance		ℓ_4 Distance		ℓ_5 Distance		ℓ_{10} Distance	
$(d = 2, \delta)$	(2,30)	(2,100)	(2,30)	(2,100)	(2,30)	(2,100)	(2,30)	(2,100)	(2,30)	(2,100)
Communication Cost [29]	0.06	0.23	0.17	4.20	3.30	–	–	–	–	–
Communication Cost (ours)	0.59	1.71	0.59	1.30	0.39	1.31	0.42	0.79	0.25	0.62
Runtime [29]	20.54	137.04	322.63	11650	9390	–	–	–	–	–
Runtime (ours)	59.93	160.75	58.09	123.97	45.89	149.81	50.21	104.23	30.11	67.81
$(d = 3, \delta)$	(3,30)	(3,100)	(3,30)	(3,100)	(3,30)	(3,100)	(3,30)	(3,100)	(3,30)	(3,100)
Communication Cost [29]	0.09	0.33	0.20	4.29	3.33	–	–	–	–	–
Communication Cost (ours)	25.24	432.01	25.68	483.70	23.57	450.39	13.11	296.49	8.93	271.74
Runtime [29]	24.52	145.62	327.83	11490	9490	–	–	–	–	–
Runtime (ours)	2090	36230	2130	39950	2020	40130	1250	28880	754.26	21190

maximal side lengths $\delta_{\max} = (\max_k \delta_{k,1}, \ldots, \max_k \delta_{k,d})$, which can be partially concealed at some efficiency cost.

Second, as briefly noted in Section 5.1, the protocol does not natively support dynamic environments with moving entities. Supporting such scenarios requires modifications to the protocol, which are sketched in Appendix E and left for future work.

6 Concrete Performance Evaluation

For comparison, we consider the state-of-the-art minimal-round fuzzy matching protocol by van Baarsen *et al.* [29]. Because their construction supports only distance-based proximity testing for ℓ_p distances, we only evaluate both protocols in this setting.

Environment. Our implementation[7] was written in Rust, and our experiments were run on a regular desktop computer with a 12th Gen, 5GHz Intel i7-12700k with 16 GB of RAM. All communication between Sender and Receiver take place over a local network (LAN), so latency is not considered in our experiments.

Setup. We experiment in six different norms ($l_\infty, l_2, l_3, l_4, l_5, l_{10}$), two different dimensions (2D, 3D) and three different radius values (30, 100, 1000). For each norm, we run the GFM protocols of Figures 4, 6 and the FM protocols of [29] through every possible combination of dimensions and radius with values of 30 and 100. For each of the experimental settings, the protocol is run 100 times and the results are averaged. Thus, results are presented as the average of 100 runs, in MB for communication and ms for time.

The ℓ_∞ case. As seen in Table 2, for both 2D and 3D cases and all chosen radii, GFM_∞ requires less time and communication for evaluation, with the gap between GFM and [29] widening as the radius is increased and as the dimension increases.

Table 2: Communication overhead Comm **and average runtime** Time **of fuzzy matching for ℓ_∞ distance.**

(d, δ)	(2,30)	(2,100)	(2,1000)	(3,30)	(3,100)	(3,1000)
Comm [29]	0.059	0.194	1.925	0.088	0.290	2.886
Comm (ours)	0.042	0.044	0.046	0.064	0.066	0.069
Time [29]	8.59	21.14	172.26	12.60	30.26	251.23
Time (ours)	8.65	8.56	9.52	11.22	11.52	12.90

[7] Code: https://github.com/John-Katis/Fuzzy-PSI-with-Fuzzy-Matching

The ℓ_p case. As seen in Table 1, for ℓ_2 and ℓ_3 distances, van Baarsen *et al.*'s protocol achieves lower communication and faster runtimes than our construction, as expected. For higher distances, however, their approach does not scale: at ℓ_4 runtimes become prohibitive, and at ℓ_5 and ℓ_{10} the protocol fails due to the exponential growth of internal lists of size δ^p, leading to memory allocation errors. Specifically, high ℓ_p values with large δ require hash tables that exceed available memory (e.g., ℓ_5 with $\delta = 100$ or ℓ_{10} with $\delta = 30, 100$), while other combinations, such as ℓ_4 with $\delta = 100$ or ℓ_5 with $\delta = 30$, would have taken excessively long to execute—up to 30 hours for 100 runs on our 16GB RAM machine. To maintain clarity and fairness, we did not report averages over fewer runs, as results would be less statistically reliable.

In contrast, our protocol remains stable across all ℓ_p distances. The geometric intuition is that ℓ_p balls vary only gradually as p increases, transitioning from circles at ℓ_2 to square-like shapes, and eventually to exact squares at ℓ_∞. Because these shapes remain similar throughout, our partitioner can approximate them consistently well. This similarity also explains why, in some cases, larger values of p even result in slightly better runtimes, as the partitioning becomes more accurate and involves larger hyperrectangles and fewer residual points when the shape approaches a square.

More complex shapes. While our benchmarks in Table 1 focus on ℓ_p distances, this choice is representative of our framework's behavior across different geometric regions. Our framework partitions any input region into hyperrectangles and residual points, which are then processed by the protocol. Therefore, the performance differences for other distances would arise primarily from variations in the number and size of hyperrectangles, while the protocol itself remains unchanged.

To further validate the approach, we implemented benchmarks for two complex 3D regions presented earlier in Appendix F.

Acknowledgments

Florias Papadopoulos is supported by armasuisse Science and Technology.

References

[1] Abhishek Bhowmick, Dan Boneh, Steve Myers, Kunal Talwar, and Karl Tarbe. 2021. The Apple PSI System. https://www.apple.com/child-safety/pdf/Apple_PSI_System_Security_Protocol_and_Analysis.pdf. Apple Technical Report.

[2] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2023. Near-Optimal Oblivious Key-Value Stores for Efficient PSI, PSU and Volume-Hiding Multi-Maps. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 301–318. https://www.usenix.org/conference/usenixsecurity23/presentation/bienstock

[3] Anrin Chakraborti, Giulia Fanti, and Michael K. Reiter. 2023. Distance-Aware Private Set Intersection. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 319–336. https://www.usenix.org/conference/usenixsecurity23/presentation/chakraborti-intersection

[4] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. 2004. Efficient Private Matching and Set Intersection. In *Advances in Cryptology - EUROCRYPT 2004*, Christian Cachin and Jan L. Camenisch (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–19.

[5] Dario Freni, Carmen Ruiz Vicente, Sergio Mascetti, Claudio Bettini, and Christian S. Jensen. 2010. Preserving location and absence privacy in geo-social networks. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management* (Toronto, ON, Canada) *(CIKM '10)*. Association for Computing Machinery, New York, NY, USA, 309–318. doi:10.1145/1871437.1871480

[6] Ying Gao, Lin Qi, Xiang Liu, Yuanchao Luo, and Longxin Wang. 2025. Efficient Fuzzy Private Set Intersection from Fuzzy Mapping. In *Advances in Cryptology – ASIACRYPT 2024*, Kai-Min Chung and Yu Sasaki (Eds.). Springer Nature Singapore, Singapore, 36–68.

[7] Gayathri Garimella, Benjamin Goff, and Peihan Miao. 2024. Computation Efficient Structure Aware PSI From Incremental Function Secret Sharing. In *Advances in Cryptology - CRYPTO 2024: 44th Annual International Cryptology Conference, CRYPTO 2024, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VIII* (Santa Barbara, CA, USA). Springer-Verlag, Berlin, Heidelberg, 309–345. doi:10.1007/978-3-031-68397-8_10

[8] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2021. Oblivious Key-Value Stores and Amplification for Private Set Intersection. In *Advances in Cryptology – CRYPTO 2021*, Tal Malkin and Chris Peikert (Eds.). Springer International Publishing, Cham, 395–425.

[9] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. 2022. Structure-Aware Private Set Intersection, with Applications to Fuzzy Matching. In *Advances in Cryptology – CRYPTO 2022, Part I (Lecture Notes in Computer Science, Vol. 13507)*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer, Cham, Switzerland, Santa Barbara, CA, USA, 323–352. doi:10.1007/978-3-031-15802-5_12

[10] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. 2023. Malicious Secure, Structure-Aware Private Set Intersection. In *Advances in Cryptology – CRYPTO 2023*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer Nature Switzerland, Cham, 577–610.

[11] Andrew Glassner. 2014. Square Packing In A Circle. https://www.imaginary-institute.com/blog/2014/07/26/square-packing-in-a-circle/ Accessed: 2025-05-29.

[12] Karl Holub, Yannick Martel, Max Kuhn, and Simon P. Couch. 2022. xrf: eXtreme RuleFit (sparse linear models on XGBoost ensembles). https://github.com/holub008/xrf. R package version 0.2.2.

[13] Kimmo Järvinen, Ágnes Kiss, Thomas Schneider, Oleksandr Tkachenko, and Zheng Yang. 2018. Faster Privacy-Preserving Location Proximity Schemes. In *CANS 18: 17th International Conference on Cryptology and Network Security (Lecture Notes in Computer Science, Vol. 11124)*, Jan Camenisch and Panos Papadimitratos (Eds.). Springer, Cham, Switzerland, Naples, Italy, 3–22. doi:10.1007/978-3-030-00434-7_1

[14] Hafsa Khalil. 2024. *Uber fined €290m for personal data transfer.* https://www.bbc.com/news/articles/cy76v561g48o Accessed: 2025-06-04.

[15] Yehuda Lindell. 2017. How to Simulate It – A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*. Springer, Cham, 293–367. doi:10.1007/978-3-319-57048-8_6

[16] Guowei Ling, Peng Tang, Fei Tang, Shifeng Sun, Shouling Ji, and Weidong Qiu. 2025. Ultra-Fast Private Set Intersection From Efficient Oblivious Key-Value Stores. *IEEE Transactions on Dependable and Secure Computing* 22, 05 (Sept. 2025), 4998–5014. doi:10.1109/TDSC.2025.3557963

[17] Andrzej Lingas, Ron Y. Pinter, Ronald L. Rivest, and Adi Shamir. 1982. Minimum Edge Length Partitioning of Rectilinear Polygons. In *Proceedings of the Annual Allerton Conference on Communication, Control, and Computing*. University of Illinois at Urbana-Champaign, Urbana, IL, USA, 53–63.

[18] Mark Manulis, Benny Pinkas, and Bertram Poettering. 2010. Privacy-Preserving Group Discovery with Linear Complexity. In *Applied Cryptography and Network Security*, Jianying Zhou and Moti Yung (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 420–437.

[19] Sergio Mascetti, Dario Freni, Claudio Bettini, X. Sean Wang, and Sushil Jajodia. 2011. Privacy in geo-social networks: proximity notification with untrusted service providers and curious buddies. *The VLDB Journal* 20, 4 (Aug. 2011), 541–566. doi:10.1007/s00778-010-0213-7

[20] Jonas Mockus. 1989. *Bayesian Approach to Global Optimization: Theory and Applications* (1 ed.). Springer, Dordrecht, Netherlands. 270 pages. doi:10.1007/978-94-009-0909-0 Originally published in Russian. Reprinted by Kluwer Academic Publishers..

[21] M. Naor and O. Reingold. 1997. Number-theoretic constructions of efficient pseudo-random functions. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*. IEEE Computer Society, USA, 458–467.

[22] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, and Dan Boneh. 2011. Location Privacy via Private Proximity Testing. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2011)*. The Internet Society, San Diego, CA, USA. https://www.ndss-symposium.org/ndss2011/privacy-private-proximity-testing-paper

[23] Ivan Oleynikov, Elena Pagnin, and Andrei Sabelfeld. 2020. Where Are You Bob? Privacy-Preserving Proximity Testing with a Napping Party. In *Computer Security - ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part I* (Guildford, United Kingdom). Springer-Verlag, Berlin, Heidelberg, 677–697. doi:10.1007/978-3-030-58951-6_33

[24] Ivan Oleynikov, Elena Pagnin, and Andrei Sabelfeld. 2022. CatNap: Leveraging Generic MPC for Actively Secure Privacy-enhancing Proximity Testing with a Napping Party. In *Proceedings of the 19th International Conference on Security and Cryptography (SECRYPT 2022)*. SCITEPRESS, Lisbon, Portugal, 237–248. doi:10.5220/0011279500003283

[25] Ivan Oleynikov, Elena Pagnin, and Andrei Sabelfeld. 2022. Outsourcing MPC Precomputation for Location Privacy. In *IEEE European Symposium on Security and Privacy Workshops (EuroS&P 2022)*. IEEE, Genoa, Italy, 504–513. doi:10.1109/EUROSPW55150.2022.00060

[26] Srinivasan Raghuraman and Peter Rindal. 2022. Blazing Fast PSI from Improved OKVS and Subfield VOLE. In *ACM CCS 2022: 29th Conference on Computer and Communications Security*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, Los Angeles, CA, USA, 2505–2517. doi:10.1145/3548606.3560658

[27] Gokay Saldamli, Richard Chow, Hongxia Jin, and Bart Knijnenburg. 2013. Private proximity testing with an untrusted server. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks* (Budapest, Hungary) *(WiSec '13)*. Association for Computing Machinery, New York, NY, USA, 113–118. doi:10.1145/2462096.2462115

[28] Pietro Tedeschi, Savio Sciancalepore, and Roberto Di Pietro. 2023. Lightweight Privacy-Preserving Proximity Discovery for Remotely-Controlled Drones. In *Proceedings of the 39th Annual Computer Security Applications Conference* (Austin, TX, USA) *(ACSAC '23)*. Association for Computing Machinery, New York, NY, USA, 178–189. doi:10.1145/3627106.3627174

[29] Aron van Baarsen and Sihang Pu. 2024. Fuzzy Private Set Intersection with Large Hyperballs. In *Advances in Cryptology - EUROCRYPT 2024: 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part V* (Zurich, Switzerland). Springer-Verlag, Berlin, Heidelberg, 340–369. doi:10.1007/978-3-031-58740-5_12

[30] Jaroslav Šeděnka and Paolo Gasti. 2014. Privacy-preserving distance computation and proximity testing on earth, done right. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security* (Kyoto, Japan) *(ASIA CCS '14)*. Association for Computing Machinery, New York, NY, USA, 99–110. doi:10.1145/2590296.2590307

[31] Laurynas Šikšnys, Jeppe Rishede Thomsen, Simonas Šaltenis, and Man Lung Yiu. 2010. Private and Flexible Proximity Detection in Mobile Social Networks. In *Proceedings of the 2010 Eleventh International Conference on Mobile Data Management (MDM '10)*. IEEE Computer Society, USA, 75–84. doi:10.1109/MDM.2010.43

[32] San-Yuan Wu and Sartaj Sahni. 1994. Fast Algorithms to Partition Simple Rectilinear Polygons. *VLSI Design* 1 (01 1994). doi:10.1155/1994/16075

A Oblivious Key Value Stores

In this section, we provide definitions and technical details for the Oblivious Key-Value Stores (OKVS) used in our protocols:

Definition 1 (Oblivious Key-Value Store). An oblivious key value store OKVS is defined over a key-space $\mathcal{K}$, a key-space $\mathcal{V}$, two security parameters κ (statistical) and λ (computational), and consists of two algorithms:

- Encode$(L; r)$: takes as input a set $L = \{(k_i, v_i)\}_{i\in[n]} \in (\mathcal{K}\times\mathcal{V})^n$ of key-value pairs and randomness $r \leftarrow\!\!\$\ \{0,1\}^\kappa$, and outputs an encoding $s \in \mathcal{V}^m$ (or, with statistically small probability, an error indicator $\perp$).
- Decode$(s, k; r)$: takes as input the encoding s, a key $k \in \mathcal{K}$ and randomness $r \leftarrow\!\!\$\ \{0,1\}^\kappa$, and outputs a value $v \in \mathcal{V}$.

Moreover, an OKVS must satisfy the following properties:

- **Correctness:** For all inputs $L \in (\mathcal{K} \times \mathcal{V})^n$ with distinct keys, and all $(k, v) \in L$:

$$\Pr[\mathsf{Decode}(\mathsf{Encode}(L), k) = v] \text{ is overwhelming..}$$

- **Obliviousness:** For any distinct key sets $\{k_i\}_i$ and $\{k'_i\}_i$, and values $v_i \leftarrow \mathcal{V}$ chosen uniformly at random, the distributions $\{\boldsymbol{s} \leftarrow \mathsf{Encode}(\{(k_i, v_i)\}_{i\in[n]})\}$ and $\{\boldsymbol{s}' \leftarrow \mathsf{Encode}(\{(k'_i, v_i)\}_{i\in[n]})\}$ are computationally indistinguishable.

Additional properties. In this work, we require the OKVS to satisfy several additional properties, namely double obliviousness $\mathbb{F}_p$-Linearity (allowing encodings to be computed through linear algebra over $\mathbb{F}_p$), satisfied by most constructions [2, 8, 26]:

- **Double obliviousness:** For any set of n distinct keys $\{k_i\}_{i\in[n]}$ and n uniformly chosen values $\{v_i\} \leftarrow\$ \mathcal{V}$, the encoding $\boldsymbol{s} \coloneqq \mathsf{Encode}(\{(k_i, v_i)\}_{i\in[n]})$ is statistically indistinguishable from a uniform element in $\mathcal{V}^m$.
- **$\mathbb{F}_p$-Linearity:** A function $\mathsf{dec} : \mathcal{K} \times \{0,1\}^\kappa \to \mathcal{V}^m$ exists such that $\mathsf{Decode}(\boldsymbol{s}, k) = \langle \mathsf{dec}(k), \boldsymbol{s} \rangle$, i.e., decoding is an inner product.

These properties imply the *independence* property, originally proved in [9] for binary OKVS, extended in [29] for the $\mathbb{F}_p$-linear case:

Lemma 2 (Independence). If an OKVS with negligible failure probability satisfies also $\mathbb{F}_p$-linearity and double obliviousness then, for any $L \coloneqq \{(k_i, v_i)\}_{i\in[n]}$ with distinct keys and any key $k \notin \{k_i\}_{i\in[n]}$, it holds that $\mathsf{Decode}(\mathsf{Encode}(L), k)$ is indistinguishable from random.

B Specialized Partitioning Algorithms

B.1 Specialized Partitioning for Polygons

Rectilinear polygons with n vertices can be partitioned into non-overlapping rectangles under various optimization goals. For minimizing the total cut length, Lingas, Pinter, Rivest, and Shamir [17] proposed a dynamic programming algorithm with $O(n^4)$ time complexity. On the other hand, for the goal of minimizing the number of rectangles, Wu and Sahni [32] designed algorithms with time complexity $O(kn)$ and $O(n \log k)$, where k is the smaller of the number of vertical and horizontal inversions of the polygon.

These exact algorithms guarantee full coverage of the polygon without leaving any residual regions. In contrast, approximate approaches may offer faster runtimes but can leave out small uncovered areas, which must then be handled separately.

B.2 Specialized Partitioning for Discs

The specialized disc partitioning procedure (Algorithm 13) for a disc $D(\boldsymbol{c}, R)$ of center $\boldsymbol{c} = (C_x, C_y)$ and radius R operates in two phases: (i) computing a rectangle partition and corresponding residual points for the origin-centered disc $D(\boldsymbol{0}, R)$, and (ii) translating these rectangles and residual points to the target center $\boldsymbol{c}$.

Since the translation is straightforward, we concentrate on the first phase. This begins by placing the largest square centered at the origin $(0, 0)$ that fits entirely within the disc of radius R. It then iteratively fills one octant of the remaining area by inserting the largest possible squares into the uncovered regions above and adjacent to existing ones. This is done by tracing lines at 45° angles from key corners and intersecting them with the disc boundary to generate new candidates using CreateSquare (Algorithm 11). Each candidate is passed to CreateRectangle (Algorithm 10), which attempts to extract the largest axis-aligned rectangle strictly contained within it; candidates that fail this test are discarded. Repeating this process yields an approximate partition of the disc into axis-aligned rectangles.[8]

After this step, the algorithm invokes ComputeResiduals (Algorithm 12) to identify residual integer points (i.e., those in the disc but outside the union of rectangles) ensuring full coverage of integer lattice points within $D(\boldsymbol{0}, R)$. Additionally, an optional fusion step can be applied to merge adjacent rectangles that are symmetric due to the partitioning process. The example in Figure 8 includes this fusion and illustrates the resulting partition.

Remark 3. Throughout the algorithms, squares and rectangles are represented as tuples $(\boldsymbol{c}, \boldsymbol{h})$, where $\boldsymbol{c} = (c_x, c_y)$ denotes the center and $\boldsymbol{h} = (h_x, h_y)$ the half-side lengths. For squares, the representation may simplify to $(\boldsymbol{c}, h)$ with $h = h_x = h_y$.

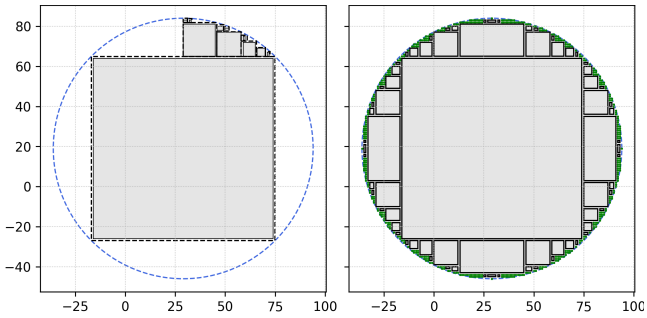


Figure 8: Illustration regarding the efficient circle partition algorithm on input radius $R = 70$ and center $(29, 19)$.

Algorithm 10 CreateRect(Sq)

Input: A square $\mathrm{Sq} = ((c_x, c_y), h)$
Output: An axis-aligned rectangle $\mathrm{Rect} = ((c'_x, c'_y), (h'_x, h'_y))$ with $c'_x, c'_y, h'_x, h'_y \in \mathbb{Z}$, or $\perp$ if no such rectangle exists

Compute $x_\ell \coloneqq \lceil c_x - h \rceil$ and $x_r \coloneqq \lfloor c_x + h \rfloor$
Compute $y_b \coloneqq \lceil c_y - h \rceil$, and $y_t \coloneqq \lfloor c_y + h \rfloor$
if $x_r - x_\ell < 1$ or $y_t - y_b < 1$ **then** ▷ ∄ Rect
  return $\perp$
end if
Set $c'_x \coloneqq \lfloor (x_\ell + x_r)/2 \rfloor$ and $h'_x \coloneqq \lfloor (x_r - x_\ell)/2 \rfloor$
Set $c'_y \coloneqq \lfloor (y_b + y_t)/2 \rfloor$ and $h'_y \coloneqq \lfloor (y_t - y_b)/2 \rfloor$

return $\mathrm{Rect} \coloneqq ((c'_x, c'_y), (h'_x, h'_y))$

C Analysis of our Fuzzy Matching Protocol for Infinity Distance

C.1 Proof of Theorem 2

On the receiver's side, we let $\{\mathrm{str}^{\mathcal{R}}_{i,j}\}_{j=1}^{J_i}$ be the representative strings output by $\mathrm{DASA}_{\mathcal{R}}$ for input w_i, δ (corresponding to the interval

[8] Note that, unlike our generic procedure, this approach does not allow precise control over the number of rectangles, but minimizes the number of residual points.

Algorithm 11 CreateSquare(P, R)

Input: A corner $\mathsf{P} = (P_x, P_y)$ of a square and the radius R
Output: A new square $\mathrm{Sq}' = ((c'_x, c'_y), h')$

Set $h' \coloneqq \left(-P_x - P_y + \sqrt{2(R^2 - (P_y - P_x)^2)}\right)/4$
Set $c'_x \coloneqq \left(3P_x - P_y + \sqrt{2(R^2 - (P_y - P_x)^2)}\right)/4$
Set $c'_y \coloneqq \left(-P_x + 3P_y + \sqrt{2(R^2 - (P_y - P_x)^2)}\right)/4$
return $\mathrm{Sq}' \coloneqq ((c'_x, c'_y), h')$

Algorithm 12 ComputeResiduals($(R, \boldsymbol{c}), H$)

Input: The radius R and center point $\boldsymbol{c} = (c_x, c_y)$ of a disc $D(\boldsymbol{c}, R)$, as well as a set of rectangles $H = \{h_1, \ldots, h_K\}$
Output: A set Res of integer points in $D(\boldsymbol{c}, R) \setminus \bigcup_k H$

Initialize two sets: H_{pts} and Res
for each rectangle $((c_x, c_y), (h_x, h_y)) \in H$ **do**
  for $x = c_x - h_x$ to $c_x + h_x$ **do**
    for $y = c_y - h_y$ to $c_y + h_y$ **do**
      Add point (x, y) to H_{pts}
    end for
  end for
end for

for $x = c_x - R$ to $c_x + R$ **do**
  for $y = c_y - R$ to $c_y + R$ **do**
    if $(x, y) \in D(\boldsymbol{c}, R)$ **and** $(x, y) \notin H_{\mathsf{pts}}$ **then**
      Add (x, y) to Res
    end if
  end for
end for

return Res

Algorithm 13 DiscPartitioner(R, center)

Input: Radius $R \in \mathbb{Z}_+$ and center point $\boldsymbol{c} = (C_x, C_y) \in \mathbb{R}^2$
Output: A set of rectangles $H \coloneqq \{\mathrm{Rect}_1, \ldots, \mathrm{Rect}_K\}$ and a set of residual points $\mathrm{Res} \coloneqq \{\boldsymbol{res}_i\}_{i=1}^{L}$

Phase 1 (Partitioning $D(\mathbf{0}, R)$):
Set $\mathrm{Sq}_0 \coloneqq ((0,0),\ R\sqrt{2}/2)$ ▷ Biggest centered square
Initialize a set $H_{\mathbf{0}}$.
Compute $h_1 \coloneqq \mathsf{CreateRect}(\mathrm{Sq}_0)$ and insert it to $H_{\mathbf{0}}$
Set $\mathrm{Sq}_1 = \left(\left(R(\sqrt{6} - \sqrt{2})/2,\ R\sqrt{3}/\sqrt{2}\right), R(\sqrt{6} - \sqrt{2})/2\right)$
Initialize a set S_{temp} and add Sq_1 to it
while $S_{\mathsf{temp}} \neq \emptyset$ **do**
  for $\mathrm{Sq} \coloneqq ((c_x, c_y), h) \in S_{\mathsf{temp}}$ **do**
    Define top left corner $\mathsf{TL} = (c_x - h, c_y + h)$
    Define bottom right corner $\mathsf{BR} = (c_x + h, c_y - h)$
    Set $(\boldsymbol{c}', h') \coloneqq \mathsf{CreateSquare}(\mathsf{TL}, R)$
    if $h' \geq 1/2$ **then**
      Add $(\boldsymbol{c}', h')$ to S_{temp}
    end if
    Set $(\boldsymbol{c}'', h'') \coloneqq \mathsf{CreateSquare}(\mathsf{BR}, R)$
    if $h'' \geq 1/2$ **then**
      Add $(\boldsymbol{c}'', h'')$ to S_{temp}
    end if
    Remove Sq from S_{temp}
    Compute $\mathrm{Rect} \coloneqq \mathsf{CreateRectangle}(\mathrm{Sq})$
    if $\mathrm{Rect} \neq \perp$ **then**
      Add Rect to $H_{\mathbf{0}}$
    end if
  end for
end while

Compute the set $\mathrm{Res}_{\mathbf{0}} \coloneqq \mathsf{ComputeResiduals}(\mathbf{0}, R), H_{\mathbf{0}})$

Phase 2 (Translation to the proper center):
Initialize empty sets H and Res
for each rectangle $\mathrm{Rect} = ((c_x, c_y), (h_x, h_y)) \in H_{\mathbf{0}}$ **do**
  Add $\mathrm{Rect}' = ((c_x + C_x, c_y + C_y), (h_x, h_y))$ to H
end for
for each point $(x, y) \in \mathrm{Res}_{\mathbf{0}}$ **do**
  Add $(x + C_x, y + C_y)$ to Res
end for

return A set of rectangles $H = \{\mathrm{Rect}_1, \ldots, \mathrm{Rect}_K\}$ and the set of residual points $\mathrm{Res} = \{\boldsymbol{res}_i\}_{i=1}^{L}$

$[w_i - \delta, w_i + \delta]$), in each $i \in [d]$). These strings are then prefixed with their dimension index and assembled into sets $R_i = \{i \| \mathsf{str}^{\mathcal{R}}_{i,j}\}_{j=1}^{J_i}$, giving the overall output of $\mathsf{RecStrings}_\infty$ for input R_{desc}:

$$R = \boldsymbol{R} = (R_1, \ldots, R_d).$$

On the sender's side, we let $\{\mathsf{str}^{\mathcal{S}}_{i,j}\}_{j=1}^{J'_i}$ be the representative strings output by $\mathsf{DASA}_{\mathcal{S}}$ for input q_i, δ in each dimension $i \in [d]$. Then, a full set of vectors $S = \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{J_{\mathcal{S}}}\}$ is formed by taking the Cartesian product of the sender's representative strings for each dimension (prefixed with their dimension index). This is the output of $\mathsf{SndStrings}_\infty$ for input $\boldsymbol{q}$, $R_{\mathsf{info}} = \{\delta, \text{“}\ell_\infty\text{”}\}$.

To proceed with the proof, we rely on the following:

Fact 1. For each dimension $i \in [d]$: (i) If $q_i \in [w_i - \delta, w_i + \delta]$, then by the correctness of DASA from [3], exactly one of the sender's strings $\{\mathsf{str}^{\mathcal{S}}_{i,j}\}_{j=1}^{J'_i}$ will match exactly one of the receiver's strings $\{\mathsf{str}^{\mathcal{R}}_{i,j}\}_{j=1}^{J_i}$; (ii) if $q_i \notin [w_i - \delta, w_i + \delta]$, no such match occurs.pr

Fact 2. Building on Fact 1, it is easy to see that, there exists a unique tuple $\boldsymbol{s} = (s_1, \ldots, s_d) \in S$ such that $\forall i \in [d], s_i \in \{i \| \mathsf{str}^{\mathcal{R}}_{i,j}\}_{j=1}^{J_i}$, only if $q_i \in [w_i - \delta, w_i + \delta]$ for all dimensions $i \in [d]$. Otherwise, no such vector exists.

Fact 3. In each dimension $i \in [d]$, the receiver generates $J_i = O(\log \delta)$ representative strings, with the exact count bounded above by $2\lfloor \log(2\delta + 1) \rfloor$. On the other hand, the sender generates exactly $J'_i = \lfloor \log(2\delta + 1) \rfloor + 1$ representative strings in each dimension $i \in [d]$, which means that the total number of vectors is:

$$J_{\mathcal{S}} = \prod_{i=1}^{d} J'_i = (\lfloor \log(2\delta + 1) \rfloor + 1)^d$$

From this point, the proof mirrors Theorems 5 and 11 of [29], splitting into two cases:

Case 1 ($\|\boldsymbol{w} - \boldsymbol{q}\|_\infty > \delta$). In this case, the sender will not get the same key to decode OKVS for at least one dimension if H_γ is a universal hash, except with $\frac{2\lfloor \log(2\delta+1) \rfloor}{2^\gamma}$ probability. After taking a union bound over each pair of points, this probability becomes $\mathsf{negl}(k)$ for $\gamma = \kappa + (d+1) \log \log \delta$.

Moreover, since the OKVS satisfies the independence property, for all $j \in [J_S]$, there exist at least one dimension $i_j \in [d]$ such that (u_{j,i_j}, v_{j,i_j}) is uniformly random over $\mathbb{G} \times \mathbb{G}$. This implies that $u^r_j = v'_j$ with probability at most $\mathsf{negl}(\kappa)$. Thus, we have inconsistent $(u_{j,i}, v_{j,i})$ pairs with $1 - \mathsf{negl}(\kappa)$ probability. Additionally, the probability that there exists some $j \in [J_S]$ such that $H_{\kappa'}(u^r_j) = v_j$ remains negligible, provided that $\kappa' = \kappa + d \log\log\delta$.

Case 2 ($\|\boldsymbol{w}-\boldsymbol{q}\|_\infty \le \delta$). In this case, we have $q_i \in [w_i-\delta, w_i+\delta], \forall i \in [d]$ and thus by Fact 2 there is a unique vector $\boldsymbol{s_j} \in S$ such that $s_{j,i} \in \{i \| \mathsf{str}^{\mathcal{R}}_{i,j}\}_{j=1}^{J_i}, \forall i \in [d]$, with overwhelming probability. Then, it is easy to see that for only this unique representative string we will have $u^r_{j,i} = v_{j,i}$ on the sender's side, which implies $u^r_j = v'_j$, $H_{\kappa'}(u^r_j) = v_j$, and thus $1 \leftarrow \mathsf{Receiver}_2$.

C.2 Proof of Theorem 3

The proof proceeds by considering separately the cases where either the sender or the receiver is corrupted:

Case 1 (Sender is corrupted). In this case, the simulator invokes $\mathcal{F}_{\mathrm{GFM}}$, getting output 1 or 0 as well as $R_{\mathsf{info}} = \{\delta, \text{“}\ell_\infty\text{”}\}$ from the nested functionality $\mathcal{F}_{\mathrm{INFO}}$, and simulates msg_1 to complete the sender's view. The simulator samples

$$L_i \coloneqq \{(\mathsf{key}_{i,j}, \mathsf{val}_{i,j})\}_j \leftarrow\$ \left(\{0,1\}^\gamma \times (\mathbb{G}\times\mathbb{G})\right)^{2\lfloor \log(2\delta+1)\rfloor}$$

for each dimension $i \in [d]$, sets $E'_i \leftarrow \mathsf{Encode}(L_i)$ and sets $\boldsymbol{E'} \coloneqq \{E'_1, \ldots, E'_d\}$. We show that the simulated $\mathsf{msg}'_1 \coloneqq (g, H \leftarrow\$ \mathbb{G}, \boldsymbol{E'})$ is computationally indistinguishable from msg_1 in the real game:

According to the DDH assumption, (g, h, h_j, h^s_j) in the real game is a DDH tuple and thus $\mathsf{val}_j \coloneqq (h_j \| h^s_j)$ is indistinguishable from a uniformly random pair of group elements. Then, according to the obliviousness of OKVS, as long as the inserted $\{\mathsf{val}_j\}$ are uniformly random (for each dimension), then E' and E are perfectly indistinguishable. Therefore, we have $\mathsf{msg}'_1 \approx_c \mathsf{msg}_1$, which is sufficient to simulate the sender's view.

Case 2 (Receiver is corrupted). In this case, since the output can be easily simulated by invoking $\mathcal{F}_{\mathrm{GFM}}$, we focus on simulating msg_2 to complete the receiver's view. First, the simulator generates

$$\{(u_j, H_{\kappa'}(v'_j))\}_{j=1}^{J_S}, \text{ where } u_j, v'_j \leftarrow\$ \mathbb{G}.$$

Then, if $1 \leftarrow \mathcal{F}_{\mathrm{GFM}}$, the simulator samples one index $J \leftarrow\$ [J_S]$ and replaces (u_J, v_J) with $(g^{r'}, H_{\kappa'}(h^{r'}))$ in the previous set, where $r' \leftarrow\$ \mathbb{Z}_q$. Otherwise, he does nothing. In the end, the simulated message would be $\mathsf{msg}'_2 \coloneqq \{(u_j, v_j)\}_{j=1}^{J_S}$. We show the simulated msg'_2 is indistinguishable from msg_2 in the real game as follows:

If $1 \leftarrow \mathcal{F}_{\mathrm{GFM}}$ then, with overwhelming probability, there exists a unique (due to Fact 2 of Appendix C.1) index J such that $u^r_{J,i} = v_{J,i}, \forall i \in [d]$, and thus $(\prod_{i=1}^d u_{J,i})^r = \prod_{i=1}^d v_{J,i}$; otherwise, for all $j \in [J_S]$ there exists at least one $i \in [d]$ such that $u^r_{j,i} \neq v_{j,i}$, and hence $(\prod_{i=1}^d u_{j,i})^r \neq \prod_{i=1}^d v_{j,i}$ with overwhelming probability. According to the random self-reduction of DDH tuples shown in Lemma 1, (g, h, u^*, v') is a randomized DDH tuple when $1 \leftarrow \mathcal{F}_{\mathrm{GFM}}$, and (u^*, v') are truly uniformly random over $\mathbb{G} \times \mathbb{G}$ otherwise. Therefore, each tuple of msg_2 is either uniformly random over $(G\{0,1\}^{\kappa'})^{J_S}$, or there is a unique DDH tuple at some location $J \in [J_S]$. Since msg_2 is shuffled, it is clear to see that $\mathsf{msg}'_2 \approx_s \mathsf{msg}_2$.

D Analysis of our Generic GFM Protocol

D.1 Proof of Theorem 5

The receiver first runs Partitioner on input a concise description $\mathcal{R}_{\mathsf{desc}} = \{\mathcal{R}_{\mathsf{priv}}, \mathcal{R}_{\mathsf{struct}}\}$ for his set $\mathcal{R}_{\mathsf{input}}$. This generates a set of K disjoint hyperrectangles $H^* = \{h_1, \ldots, h_k\}$, each described by their centers $\boldsymbol{c}_k$ and half-side lengths in each dimension $\boldsymbol{\delta}_k$, as well as a set of residual points $\{\mathsf{res}_\ell\}_{\ell=1}^L$. Define the set

$$H_{\mathsf{int}} \coloneqq \{\boldsymbol{a} \mid \boldsymbol{a} \in \mathbb{Z}^d \text{ and } \boldsymbol{a} \in h_k, \text{ for some } k \in [K]\}.$$

By construction of the Partitioner: $\mathcal{R}_{\mathsf{input}} = H_{\mathsf{int}} \cup \{\mathsf{res}_\ell\}_{\ell=1}^L$. We proceed with the proof by splitting it into two cases:

Case 1 ($\boldsymbol{q} \notin \mathcal{R}_{\mathsf{input}}$). In this case, we have $\boldsymbol{q} \notin h_k, \forall k \in [K]$ and $\boldsymbol{q} \notin \{\mathsf{res}_\ell\}_{\ell=1}^L$, and thus the sender will not get the same key to decode for at least one dimension on any OKVS E_i, if H_γ is a universal hash function, with probability $2\lfloor \log(2\delta+1)\rfloor / 2^\gamma$. Similarly, he will not be able to find a correct key to decode E_{res} if H_γ is a universal hash function, with probability $L/2^\gamma$. After taking a union bound on all possibilities, it is $\mathsf{negl}(\kappa)$.

Moreover, this results in inconsistent $\{(u_t, v_t)\}_{t=1}^{KJ_S+1}$ pairs with $1 - \mathsf{negl}(\kappa)$ probability, as in Theorem 2, and the probability that there exists some $t_* \in [KJ_S + 1]$ such that $H_{\kappa'}(u^r_{t_*}) = v_{t_*}$ is still negligible if $\kappa' = \kappa + \log K + d \log\log\delta_{\max}$.

Case 2 ($\boldsymbol{q} \in \mathcal{R}_{\mathsf{input}}$). By our previous analysis, we have that either $\boldsymbol{q} \in h_k$, for some $k \in [K]$, or $\boldsymbol{q} \in \{\mathsf{res}_\ell\}_{\ell=1}^L$ (since the sets are disjoint). If $\boldsymbol{q} \in h_k$, for some $k \in [K]$, then correctness reduces to Theorem C.1, being true with overwhelming probability for universal hash functions H_γ and $H_{\kappa'}$. If on the other hand, $\boldsymbol{q} \in \{\mathsf{res}_\ell\}_{\ell=1}^L$, then again it is easy to see that correctness reduces to a simpler variant of Theorem C.1 since $\mathsf{GetList}_{\mathsf{res}}$ and $\mathsf{GetTuple}_{\mathsf{res}}$ are just simpler variants of GetList and GetTuple respectively.

D.2 Proof of Theorem 6

The proof proceeds by considering separately the cases where either the sender or the receiver is corrupted:

Case 1 (Sender is corrupted). In this case, the simulator invokes $\mathcal{F}_{\mathrm{GFM}}$, getting output 1 or 0 as well as

$$R_{\mathsf{info}} \coloneqq (\boldsymbol{\delta}_{\max} \coloneqq (\delta'_1, \ldots, \delta'_d), \mathcal{R}_{\mathsf{struct}}, \{\mathsf{label}_k\}_{k=1}^K, \mathsf{label}_{\mathsf{res}})$$

from the nested functionality $\mathcal{F}_{\mathrm{INFO}}$, and simulates msg_1 to complete the sender's view. Before proceeding, we remark that the information available to the sender through R_{info} is accounted for both in the ideal functionality and the real protocol, and therefore does not constitute leakage. Intuitively, this information reveals the structure of the receiver's input region, but not its location in space (which is acceptable for all distance-based proximity testing applications, and most region-based ones).

Now, in order for msg_1 to be simulated, the simulator invokes Partitioner with the information gotten from $\mathcal{R}_{\mathsf{struct}}$, in order to find the half-side lengths $\{\boldsymbol{\delta}_k\}_{k=1}^K$ for each hyperrectangle of the partition, as well as the number of residual points L (or, the protocol can

be modified so that the receiver explicitly sends this information). Then, the simulator can sample

$$L_{k,i} \coloneqq \left\{(\mathsf{key}_{k,i,j}, \mathsf{val}_{k,i,j})\right\}_j \leftarrow\$ \left(\{0,1\}^\gamma \times \mathbb{G}^2\right)^{2\lfloor \log(2\delta_{k,i}+1)\rfloor},$$

for each dimension $i \in [d]$ and hyperrectangle index $k \in [K]$, and set $E'_i \leftarrow \mathsf{Encode}(L_{1,1} \cup \cdots \cup L_{k,i})$. Moreover, the simulator can sample

$$L_{\mathsf{res}} \coloneqq \left\{(\mathsf{key}_l, \mathsf{val}_l)\right\}_l \leftarrow\$ \left(\{0,1\}^\gamma \times \mathbb{G}^2\right)^L,$$

and set $E'_{\mathsf{res}} \leftarrow \mathsf{Encode}(L_{\mathsf{res}})$. Utilising the above, the simulator sets

$$\mathsf{msg}'_1 \coloneqq (g, h \leftarrow\$ \mathbb{G}, R_{\mathsf{info}}, \mathbf{E}, E_{\mathsf{res}})$$

Then, according to Theorem 3, $E'_i \approx_c E_i$ and similarly $E'_{\mathsf{res}} \approx_c E_{\mathsf{res}}$. Thus, $\mathsf{msg}'_1 \approx_c \mathsf{msg}_1$.

Case 2 (Receiver is corrupted). If the receiver is corrupted, the simulator needs to simulate msg_2 to complete the receiver's view. To this end, the simulator invokes the ideal functionality to get 1 or $0 \leftarrow \mathcal{F}_{\mathrm{GFM}}$. The simulator generates the set

$$\{(u_j, H_{\kappa'}(v'_j))\}_{j=1}^{KJ_S+1}, \text{ where } u_j, v'_j \leftarrow\$ \mathbb{G}.$$

If $1 \leftarrow \mathcal{F}_{\mathrm{GFM}}$, the simulator samples one index $J \leftarrow\$ [KJ_S + 1]$ and replaces (u_J, v_J) with $(g^{r'}, H_{\kappa'}(h^{r'}))$ in the previous set, where $r' \leftarrow\$ \mathbb{Z}_q$. Otherwise, he does nothing. The simulated message would be $\mathsf{msg}'_2 \coloneqq \{(u_j, v_j)\}_{j=1}^{J_S}$. After this, the proof works similarly to the one in Appendix D.2, allowing us to show that $\mathsf{msg}'_2 \approx_c \mathsf{msg}_2$, which completes the proof.

D.3 Proof of Theorem 7

First, we note that both DASA and Partitioner can be excluded from the computational complexity analysis, as they are treated as preprocessing steps in our applications. In particular, Partitioner can generate accurate partitions for geometric structures without knowledge of the receiver's specific region. For example, a polytope anchored at fixed public positions in $\mathbb{R}^3$ can be partitioned once, and the receiver can apply a translation to the resulting hyperrectangle centers and residual points to align with their actual location.

Moreover, similar to Theorem 4, the complexity bounds hold assuming that OKVS supports constant rate, linear-time encoding, and constant-time decoding. We also note that, while the sender may not know δ in advance, they can still precompute keys up to an expected δ or even for all possible values up to $\mathsf{MaxBitLen} - 1$, in order to save online costs.

We compute these complexities by leveraging the fact:

$$\delta_{\max} \coloneqq \max_i \delta_{\max,i} \geq \max_k \delta_{k,i}.$$

This simplification yields an upper bound that overestimates the true complexity, making our bounds conservative.

E Addressing Limitations

Concealment of Structural Information. As noted in Section 5, the GFM functionality can be adapted so that the sender receives only the maximal side lengths $\boldsymbol{\delta}_{\max}$. To further obscure the receiver's structure, dummy data can be added while increasing K to $K' > K$ and $\boldsymbol{\delta}_{\max}$ to $\boldsymbol{\delta}'_{\max} > \boldsymbol{\delta}_{\max}$ without affecting correctness.

However, we chose not to adopt this more concealed functionality for three reasons. First, even with these modifications, the sender would still gain some information via K' and $\boldsymbol{\delta}'_{\max}$, and quantifying the resulting privacy depends heavily on the specific application and execution context. Second, increasing K' and $\boldsymbol{\delta}'_{\max}$ comes at the cost of reduced protocol efficiency. Third, in certain applications, the sender may require knowledge of the exact thresholds and structure being queried to opt to participate.

Support for Dynamic Environments. For scenarios involving moving entities, the protocol can be extended by equipping the receiver with a guard region that covers its possible displacement during the protocol's runtime, while the sender is treated as fixed or short-term predictable. This approach parallels techniques used in related proximity-detection work [28] and preserves the core protocol design. More refined variants may adapt the guard region to motion estimates or communication delays, enabling more accurate behavior in highly dynamic environments.

F Additional Benchmarks

We present benchmarks for two complex 3D regions presented earlier in the paper. The first is the union of two spheres with centers $\boldsymbol{c}_1 = (15, 26, 42)$ and $\boldsymbol{c}_2 = (50, 42, 17)$, both with radius 30. The second is the shopping mall model composed of multiple axis-aligned hyperrectangles: main hall $(0,0,0)$–$(100,50,15)$ split into three floors; left wing $(-40,10,0)$–$(0,40,10)$; right wing $(100,10,0)$–$(140,40,10)$; food court $(20,50,0)$–$(80,70,5)$; cinema $(20,50,5)$–$(80,70,12)$; atrium $(40,20,0)$–$(60,30,15)$; elevator shaft $(48,24,0)$–$(52,28,15)$; and roof facility $(90,0,15)$–$(100,10,20)$.

Each region was benchmarked following the methodology used in the main experiments: a single execution of Partitioner, followed by 100 runs of the main protocol with random input points. Reported values are averages over these runs. In Table 3, Partitioner *Time* reflects one run of the partitioning algorithm; *GFM Time* is the average fuzzy matching time; and *Total Comm. Cost* is the corresponding communication overhead.

Table 3: Total communication cost, and execution times for Partitioner **and GFM.**

Complex Shape	Partitioner Time	GFM Time	Total Comm.
Union of Spheres	3.91 s	4.71 s	54.08 MB
Shopping Mall	363.76 ms	285.32 ms	1.37 MB

Received 26 August 2025; revised 4 December 2025; accepted 10 December 2025