Privately Evaluating Region Overlaps with Applications to Collaborative Sensor Output Validation

Anrin Chakraborti
Duke University

Michael K. Reiter
Duke University

***Abstract*—Advances in computer vision have made it possible to accurately map objects as regions in 3-dimensional space using LIDAR point clouds. These systems are key building blocks of several emerging technologies including autonomous vehicles. Comparing and validating the output of sensors at different vantage points observing the same scenery can enable these systems to detect faults, identify common obstacles, and improve decision making. However sharing sensor outputs among mutually untrusting parties can leak unwanted information, e.g., model parameters or relative location of the sensors. This work initiates the study of cryptographic protocols that enable two parties observing regions (or objects) in an arbitrary-dimension Euclidean space to privately detect if the regions overlap and approximate the volume of the overlapping region. The protocols rely only on cheap symmetric-key primitives and feature reasonable communication costs and compute times. As applications, the protocols have been benchmarked on data generated from the CARLA autonomous driving simulator and the ScanNet 3D image dataset; they outperform a 2PC garbled-circuit baseline in communication volume and compute time. For instance it takes roughly 0.5 seconds to approximate the volume of the overlapping region of two 3D boxes with low error probability.**

1. Introduction

The task of visualizing and identifying objects in the surrounding environment is an essential part of computer vision systems. Autonomous vehicles that rely on sensors (e.g., LIDAR) to visualize objects are prominent applications of these systems. However, faulty sensors can have catastrophic consequences, e.g., when a vehicle fails to detect an obstacle in its path. Sensor fusion and other collaborative techniques in artificial intelligence have shown that when multiple parties validate each others' findings they can detect faults and make decisions with less uncertainty [10, 35]. This concept can be very useful when applied to vision systems for autonomous vehicles. Consider the following two example applications:

- **Connected Autonomous Vehicles (CAVs)** [9, 11, 19, 27, 40] is an emerging technology that enables autonomous vehicles to inter-connect and share information. CAVs have positive influences on traffic flow [39], environmental impact [26, 36] and road safety [9, 44]. In fact, the NHSTA predicts that vehicle-to-vehicle communication will potentially reduce or eliminate a significant number of road crashes [1, 9]. In CAVs, sensed information such as regions containing traffic lights, traffic signs, and on-road movement can be shared among vehicles [19].
- **Cooperative Obstacle Detection** enables autonomous vehicles to detect obstacles by sharing information with each other. There are proposals for using this technology for military vehicles including underwater autonomous vehicles (UAVs) [20]. Cooperation can enable vehicles that have limited fields of view, due to terrain or by virtue of being covert, to detect obstacles/targets in path with the aid of information obtained from other vehicles.

However, if the parties do not fully trust each other, the task of sharing information without leaking sensitive information becomes challenging. For instance, vehicles from different manufacturers may use proprietary models to visualize the environment, and revealing the output(s) may allow other parties to infer information about the models. Similarly, a submarine may not want to reveal its entire set of sensed objects, lest this reveal too much about its precise location in the environment.

To tackle these problems, we initiate the study of private protocols that detect if the regions sensed in space by two parties overlap and estimate the volume of overlap. We are mainly concerned with Euclidean spaces of arbitrary dimensions. We ask the following question:

> *Suppose two parties each has a set of regions, where each region is defined by a set of points in Euclidean space. How can each party determine the regions in its set that overlap with one or more regions in the other party's set, while revealing nothing more about its own set to the other party?*

This task has several technical challenges that remain unaddressed with existing tools. Cryptographic tools for private matching, e.g., private equality testing and private set intersection, detect exact matches of elements; this will not suffice to solve our problem since sensor outputs can include noise. Fuzzy protocols [4, 14, 41] can alleviate this problem, but they are designed for point comparisons. When comparing regions which comprise (possibly infinite) sets of such points, these techniques do not scale. Consider the problem of detecting if two polygons overlap. Protocols that compute similarity based on Euclidean distance (e.g., [14]) can detect when points of one polygon lie inside the other by computing pairwise distances between the point and vertices of the other polygon. However, it is not possible to repeat this process for all the points as there may be infinitely many. Thus, to

detect such overlaps, the protocol needs to compute over geometric primitives such as edges or faces of objects, which is missing with current fuzzy matching protocols.

There is also limited work on privacy-preserving protocols for accomplishing computational geometry tasks in 2D, e.g., detecting intersections of convex 2D polygons [2]; however it is not obvious how these tools can be extended to higher dimensions and at what costs. Geometric tasks that are relatively simple in 2D often become exponentially more expensive in higher dimensions. Consider for example the task of computing the area of overlap of two polygons in 2D. There are several protocols that realize this function, even in a private setting [2, 30]. The protocols are straightforward since the overlapping region of two polygons can be determined by computing the intersections of the edges of the polygons. However, the task is more complex in higher dimensions where computing the overlapping region is non-trivial [3, 17].

Second, in our use-case the protocols will be executed between resource-constrained devices, e.g., mobile or edge-computing devices. Thus, compute-intensive tools like fully homomorphic encryption, while being suitable for implementing private versions of existing protocols that perform this task non-securely, are too expensive. Instead, we would like to rely on fast(er) primitives.

We answer our question affirmatively with protocols for overlap detection and volume of overlap measurement for convex regions. The protocols rely on efficient primitives and leverage the following key ideas.

Bounding Box Overlaps: The first step in detecting if two arbitrary (convex) regions overlap is to detect if their corresponding *bounding boxes* overlap. A bounding box is a simple geometric approximation of any complex shape (e.g., a 3D LIDAR point cloud) and is typically a rectangular cuboid of the smallest dimension that surrounds the original region. If the bounding boxes do not intersect, the original regions definitely do not intersect, as well. If the bounding boxes overlap, more fine-grained protocols are required to detect an overlap with certain guarantee.

We present two private protocols for detecting overlaps of bounding boxes in a d-dimensional space. Our first protocol takes as input axis-aligned bounding boxes, i.e., cuboids whose edges are aligned with the canonical axes, and determines if the boxes overlap. For this, the protocol computes the *Minkowski difference* of the two cuboids and checks if the origin lies inside the set of points in the Minkowski difference. The communication cost of the protocol scales linearly in d, and computing the Minkwoski difference only involves d subtractions which can be easily realized in a secure computation framework.

Our second protocol takes as inputs two arbitrarily aligned cuboids. The protocol is based on detecting if the edges of one cuboid intersect the faces of the other cuboid.

Approximate Volume of Overlapping Region: Detecting if two regions overlap is not enough to correlate and verify the outputs of two sensors: while the sensors may detect the same object, their relative positioning may be incorrect. Consider the toy example in Fig. 1. Both cars fitted with sensors detect a tree on the road, but sensor 2 places it in an incorrect location. Detecting this as a match is an incorrect conclusion in this case.

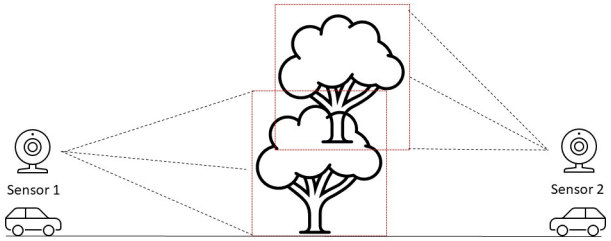


Figure 1: Two sensors detecting the same object but with incorrect relative positioning. Detecting intersection on its own does not indicate that sensor 2 is faulty.

One way to mitigate this problem is by additionally measuring the *amount of overlap* between the two regions. Our assumption is that if the volume of the overlapping region is large, then the regions are likely the same object.

We present two protocols that privately approximate the volume of overlapping regions of arbitrary convex polytopes. Our protocols build on the idea of approximating the volume of the overlapping region (between tunable lower and upper bounds) of two regions [3]. The key technical challenge is privately determining when a point inside one region also lies within the other region. For arbitrary polytopes in our first protocol, we solve the problem by leveraging the fact that a point lies within a convex polytope if it is on the same side of all the faces of the polytope. Mathematically, this implies that the dot product of the inward normals of all the faces of the polytope with the point vector should be less than zero. We check this condition using a technique combining oblivious linear evaluation and limited use of a generic secure 2PC circuit that solves the millionaire's problem.

We provide an optimization in our second protocol when the polytope in question can be easily triangulated into simplicial structures, e.g., cuboids or polyhedrons. The mathematical property we leverage is that for any d-dimensional simplex, the barycentric coordinates of any interior point lies in the range $[0, 1]$. We implement this check using a combination of oblivious linear evaluation and a range checking protocol that can be implemented either as a garbled circuit, or with tailor-made protocols [4].

Implementation: We have implemented our protocols and benchmarked them against 2PC garbled circuit baselines, and the protocols of Atallah and Du [2] for polygons extended to arbitrary-dimension polytopes. We have used a combination of real world and randomly generated datasets to benchmark the protocols. This includes a dataset of 3D axis-aligned bounding boxes of road signs, vehicles, etc. generated from the CARLA autonomous driving simulator, and 3D oriented bounding boxes of objects in the ScanNet dataset. Our protocol detects overlap of 3D axis-aligned boxes generated from the CARLA simulator $4.2\times$ faster than the garbled circuit baseline, with lower communication requirements. Approximating the volume of overlap of two bounding boxes with very low error probability (0.001) requires 508 ms.

Our protocol detects overlaps of 3D oriented boxes 1.4–6.1$\times$ faster than the baseline. Similarly, approximating the volume of overlap is up to $4.17\times$ faster than the baseline with lower communication requirements. In all cases, we outperform the protocols of Atallah and Du [2].

2. Related Work

Functionality	This Work	Atallah and Du [2]
Point Inclusion (general)	$O(\phi d\lambda)$	$O(\phi d\lambda\kappa)$
Point Inclusion (simplex)	$O(d^2\lambda)$	$O(d^2\lambda\kappa)$
Overlap Detection (cube)	$O(d\lambda)$	$O(2^d\lambda\kappa)$
Overlap Deetction (general)	$O(\beta\phi\lambda)$	$O(\beta\phi\lambda\kappa)$
Overlap Volume (cube)	$O(d\lambda)$	-
Overlap Volume (simplex)	$O(d^2\lambda)$	-
Overlap Volume (general)	$O(\phi d\lambda)$	-

TABLE 1: Asymptotic comparison of our protocols with the protocols proposed for 2D by Atallah and Du [2]. ϕ: number of faces, β: number of vertices, d: dimensions, λ: computational security parameter, κ: key size for the additively homomorphic scheme used in the protocols of Atallah and Du [2].

There is extensive work on cryptographic protocols that privately compare and match items [13, 18, 23, 24, 32, 33, 34]; however none of these techniques extend trivially to matching solid objects/regions in Euclidean space.

Comparison with Atallah and Du [2]: There is limited work on two-party private protocols for computing tasks related to solid geometric shapes and regions in Euclidean space. The most prominent is the work by Atallah and Du [2]; our protocols bear some resemblance to this work. Indeed, the geometric principles underlying both are valid in arbitrary-dimension Euclidean spaces. However, our protocols are built from cheaper primitives: all our protocols rely on oblivious linear evaluations with minimal use of garbled circuits, while the protocols of Atallah and Du [2] use additively homomorphic encryption and garbled circuits. Specifically, their protocols rely on a protocol to privately compute the scalar product of two vectors by splitting the vectors into λ shares and computing over the corresponding encrypted ciphertexts using a homomorphic encryption scheme. Thus, the communication cost scales linearly in λ, and the ciphertext size, κ. We circumvent the need for secret-sharing and the use of additively homomorphic encryption for private scalar product computation by using OLEs (see Sec. 5); thus, the communication costs of our protocols are independent of κ (see Table 1). Additionally, the use of OLEs over additively homomorphic encryption makes our protocols faster in practice.

Privacy-Preserving Proximity Detection: A loosely related task to detecting overlaps (as proposed in this work) in 2D is privacy-preserving proximity detection [29, 30, 47], wherein a trusted user of a service (say Alice) wants to learn if another user known to her (say Bob) is currently nearby. However, current techniques only work in 2D Euclidean spaces and extending these ideas even to 3D while ensuring similar privacy goals is non-trivial. In contrast, our goal is to design protocols for any arbitrary dimension, and particularly applying them to 3D spaces.

3. Background

3.1. Geometric Properties

This section presents the necessary notations and background for the geometric concepts we will extensively use in our technical description. The definitions will only pertain to the details necessary to understand the protocols, and in some cases will be presented relatively informally. More complete descriptions can be found in [6, 7].

Position Vector of a point $pt = (x_1, \ldots, x_d) \in \mathbb{R}^d$ is its relative position to an arbitrary reference point. Let $pt' = (x'_1, \ldots, x'_d) \in \mathbb{R}^d$ be an arbitrary reference point. Then, the position vector of pt with reference to pt' is the straight line segment from pt' to pt. The vector corresponding to this line segment, denoted henceforth as $\vec{pt}$, is derived as follows: for each $i \in [1, d], \vec{pt}[i] = x'_i - x_i$. When the position vector is defined in reference to the origin of the axes $(0, \ldots, 0)$, we denote it as $\vec{pt} = [x_1, \ldots, x_d]$.

Normal Vector of a surface is a vector that is perpendicular to the surface. For a non-curved surface, all (infinitely many) normal vectors are parallel to each other.

- For a line segment $\mathcal{L}$ with end points (x_1, y_1) and (x_2, y_2), let $\vec{\mathcal{L}} = [x_2 - x_1, y_2 - y_1]$ be the vector corresponding to the line segment. Then, facing in the direction that $\vec{\mathcal{L}}$ points to, we may compute normal vectors that are perpendicular to $\mathcal{L}$ and point to the space on the left of the line segment, and similarly normal vectors that are perpendicular to $\mathcal{L}$ and point to the space on the right. These normal vectors are computed as $\vec{n} = [y_1 - y_2, x_2 - x_1]$ and $-\vec{n} = [y_2 - y_1, x_1 - x_2]$ and they are in opposing directions. To check if $\vec{n}$ points to the left (and resp. to the right), we can select any point that lies on the left of the line segment, say pt with position vector $\vec{pt}$ and compute $\vec{pt} \cdot \vec{n}$. If $\vec{pt} \cdot \vec{n} \geq 0$, then $\vec{n}$ points to the space on the left of the line segment.
- For the face of a polytope, a normal vector is computed by taking the cross-product of the vectors corresponding to any two *non-parallel* edges of the face. The inward normal vector of a face of a polytope is a vector perpendicular to the face that points in the direction of the space lying inside the polytope. Whether a normal vector is inward facing can be determined as above by computing the dot product of the normal vector with the position vector of a random point lying inside the polytope and checking the sign of the output.

d-Polytope is a d-dimensional object with flat faces. For example, a polygon is a 2-dimensional polytope, and a polyhedron is a 3-dimensional polytope. In this work we are concerned with convex polytopes, which may be considered as the convex hull of a finite set of points (vertices). This definition inherently implies that the polytope is bounded, i.e., it has finite volume.

Fact 1. *Let $\mathcal{P}$ be a d-polytope and let $\{f_1, \ldots, f_\phi\}$ be the set of its faces. Let $\{\vec{f_1}, \ldots, \vec{f_\phi}\}$ be the inward facing normal vectors of the faces, i.e., vectors perpendicular to the faces but facing towards the interior of $\mathcal{P}$. Also, let $p_1, \ldots, p_\phi$ be points on the boundary of the polytope such that p_i is an arbitrary point on f_i. Let $pt = (x_1, \ldots, x_d) \in \mathbb{R}^d$ be a point and $\vec{pt_i}$ be its position vector with respect to p_i. Then, $pt \in \mathcal{P}$ iff $\vec{f_i} \cdot \vec{pt_i} \geq 0$ for all $i \in [1, \phi]$,*

Fact 1 implies that if we compare the vector drawn from any arbitrary point on a face of a polytope to a point pt, with the inward facing normal vector of the face, then both these vectors should face towards the half space that lies in the interior of the polytope if pt is inside the polytope. For this, we check if the angle between these

vectors is in $[0, \pi/2]$, which implies that the dot product of these two vectors is ≥ 0. This process is repeated for all faces to ensure that the point lies inside the polytope.

d-Cuboid is a polytope where each face is a quadrilateral. We will focus on rectangular cuboids where each face is a rectangle, and opposite faces are congruent rectangles [6]. Rectangular cuboids are the most common type of bounding boxes used in overlap detection tasks. A d-cuboid is represented by the coordinate of the top left corner, $(x_1, \ldots, x_d)$, and the lengths along each of the d axes $\ell_1, \ldots, \ell_d$.

d-Simplex is the *simplest* d-dimensional convex polytope. For instance the 2-simplex is a triangle, while the 3-simplex is a tetrahedron. More formally, the d-simplex is the convex hull of $d+1$ *affinely independent* points $\{v_1, \ldots, v_{d+1}\}$ which implies that $v_1 - v_{d+1}, v_2 - v_{d+1} \ldots, v_d - v_{d+1}$ are *linearly independent*. The d-simplex is then defined as the set of points

$$S := \left\{ \theta_1 v_1 + \ldots + \theta_{d+1} v_{d+1} \;\middle|\; \sum_{i=1}^{d+1} \theta_i = 1 \wedge \forall i : \theta_i \in \mathbb{R}_{\geq 0} \right\}$$

where $\mathbb{R}_{\geq 0}$ denotes the nonnegative reals.

Barycentric Coordinate System is a coordinate system for d-dimensional Euclidean space in which the location of a point is defined relative to a d-simplex.

Definition 1 (Barycentric Coordinate)**.** *Consider a d-simplex, $\mathcal{S}$ with set of vertices $\{v_1, \ldots, v_{d+1}\}, v_i \in \mathbb{R}^d$. Let $\{\vec{v}_1, \ldots, \vec{v}_{d+1}\}$ be the set of position vectors of the vertices of $\mathcal{S}$. Given any $pt \in \mathbb{R}^d$ and its corresponding position vector $\vec{pt}$, there always exists a unique $(a_1, \ldots, a_{d+1}) \in \mathbb{R}^{d+1}$ such that*

$$(a_1 + \ldots + a_{d+1})\ \vec{pt} = a_1 \vec{v}_1 + \ldots + a_{d+1} \vec{v}_{d+1}$$

$(a_1, \ldots, a_{d+1})$ is the barycentric coordinate of pt.

Fact 2. *For a d-dimensional point $pt \in \mathbb{R}^d$ and a d-simplex $\mathcal{S}$ defined by its vertices $\{v_1, \ldots, v_{d+1}\}$, let $(a_1, \ldots, a_{d+1})$ be the barycentric coordinate of pt relative to $\mathcal{S}$. Then, $pt \in \mathcal{S}$ iff $\forall i \in [1, d+1], a_i \in [0, 1]$.*

Definition 2. *Given two sets $\mathcal{C}_1$ and $\mathcal{C}_2$ comprising position vectors of all the points that lie inside or on the boundary of two convex bodies in $\mathbb{R}^d$, the Minkowski sum $\mathcal{C}_1 \oplus \mathcal{C}_2$ and difference $\mathcal{C}_1 \ominus \mathcal{C}_2$ are defined as*

$$\mathcal{C}_1 \oplus \mathcal{C}_2 := \left\{ \vec{x} + \vec{y} \;\middle|\; (\vec{x}, \vec{y}) \in \mathcal{C}_1 \times \mathcal{C}_2 \right\}$$
$$\mathcal{C}_1 \ominus \mathcal{C}_2 := \left\{ \vec{x} - \vec{y} \;\middle|\; (\vec{x}, \vec{y}) \in \mathcal{C}_1 \times \mathcal{C}_2 \right\}$$

Both $\mathcal{C}_1 \oplus \mathcal{C}_2$ and $\mathcal{C}_1 \ominus \mathcal{C}_2$ are the position vectors of points that lie inside or on the boundary of a convex body in $\mathbb{R}^d$.

Fact 3. *For two convex d-polytopes $\mathcal{P}_1$ and $\mathcal{P}_2$ represented by the set of points $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, the two polytopes overlap iff the origin is in $\mathcal{C}_1 \ominus \mathcal{C}_2$.*

Separating Axis Theorem (SAT). Informally, the separating axis theorem states that two polytopes do not overlap iff there is a separating axis between them which is either perpendicular to a face of one of the polytopes or is perpendicular to an edge of either of the two polytopes [17]. An axis is a separating axis for two sets of points if the projections of the sets of points on the axis are disjoint. For d-dimensional cuboidal boxes, SAT checks $\mathrm{O}(d^2)$ axes.

$\mathcal{F}_{ole}$: Ideal Function for Oblivious Linear Evaluation (OLE)

Parameters: Parties Alice and Bob, and finite field $\mathbb{F}$ from which inputs are drawn.

Inputs: Alice has input $x \in \mathbb{F}$ and Bob has as input a pair $(u, v) \in \mathbb{F}$.

Output: Alice learns $z = ux + v$. Bob learns $\perp$.

$\mathcal{F}_{vole}$: Ideal Function for Vector OLE (VOLE):

Parameters: Parties Alice and Bob, and finite field $\mathbb{F}$ from which inputs are drawn.

Inputs: Alice has input $x \in \mathbb{F}$ and Bob has as input a pair of vectors $(\vec{u}, \vec{v}) \in \mathbb{F}^d \times \mathbb{F}^d$.

Output: Alice learns $\vec{z} = \vec{u}x + \vec{v}$. Bob learns $\perp$.

Figure 2: Ideal functions for oblivious linear evaluation (OLE), and vector oblivious linear evaluation (VOLE)

TABLE 2: Table of notation

$\mathbb{R}^d$	d-dimensional Euclidean space
$\vec{pt}$	position vector of $pt \in \mathbb{R}^d$
$\mathcal{S}$	d-dimensional Simplex
$\mathcal{P}$	d-dimensional polytope
$\mathcal{B}$	d-dimensional cuboidal box
$\mathcal{C}$	d-dimensional convex object
Vol_{int}	volume of overlap of two d-dimensional convex bodies
Vol_{apprx}	approximate volume of overlap of two d-dimensional convex bodies

3.2. Cryptographic Primitives

Cryptographic Notation: We use the following standard notations: $\mathbb{F}$ is a finite field with $|\mathbb{F}| = \mathrm{O}(2^{\lambda})$, and λ is a security parameter. $\mathbf{negl}(\cdot)$ is a function that is negligible in the input parameter; e.g., $\mathbf{negl}(\lambda) = \mathrm{O}(2^{-\lambda})$.

Oblivious Linear Evaluation (OLE): Oblivious linear evaluation (OLE) is a two-party cryptographic primitive wherein Alice inputs $x \in \mathbb{F}$; Bob inputs $u, v \in \mathbb{F}$; and Alice obtains $ux + v$ *without learning* u *and* v.

Vector Oblivious Linear Evaluation (VOLE): Vector OLE (VOLE) is an extension of the OLE functionality, where Bob's input is a pair of vectors, and Alice learns a linear combination of the vectors. Fig. 2 describes the VOLE functionality. The state-of-the-art VOLE protocol [42] is based on the learning parity with noise (LPN) assumption. Further technical details can be found in [42].

Garbled Circuit: Yao's garbled circuit [28, 43] is a generic tool for secure two-party computation which enables two parties, a circuit *generator* and *evaluator* jointly compute a function that is encoded as a boolean circuit. There is extensive research on optimizing garbled circuit constructions and their applications [25, 38, 46]. We will make black box use of existing garbled circuit constructions, and in particular circuits that solve the *millionaire's problem* where one party holds a value a and the other party holds a value b, and they compute whether $a > b$ (and resp. $b > a$) without revealing the values.

4. Security Definitions

Parties: We assume that two *semi-honest* (a.k.a. honest-but-curious) mutually untrusting parties Alice and Bob run

the protocols. The parties may learn information from the intermediate results but do not deviate from the protocol.

Assumptions: We make the following assumptions:

(1) **Finite Space:** The Euclidean space under consideration for all protocols is bounded, and the parties know the end-points along all of the axes. This is reasonable since compute devices at a particular location only have a limited and finite visibility, and can sense points in the space with only finite precision.

(2) **Common Frame of Reference:** Parties have the same frame of reference for a d-dimensional Euclidean space in $\mathbb{R}^d$; i.e., the origin in Alice and Bob's views of $\mathbb{R}^d$ coincide, and the d axes are correspondingly aligned[1]. So, any region is represented the same by each of them. We also assume (for simplicity) that all points, geometric objects, etc., that we consider reside in a space with positive coordinates; e.g., when considering a 2-dimensional space, we assume that all points, objects are in the first quadrant.

(3) **Computational Assumptions:** Since the protocols are described are over $\mathbb{R}^d$, we need to deal with coordinates in $\mathbb{R}$. Since the space is finite, the fixed point representations of the coordinates in the space can be mapped to a finite field. We assume (unless stated otherwise) that the coordinates used in our protocols are fixed-point representations of real numbers. Even without fixed point representations there are techniques to map rational numbers to finite fields [5, 12]. However, since such techniques are not a contribution of this work, we omit these details here.

(4) **Convex Regions:** The (two-party) protocols we will describe are primarily designed for convex objects. Nonetheless, our protocols can be extended to non-convex objects using standard techniques like convex decomposition[2] (wherever applicable). This transformation comes at the cost of additional pre-processing that is performed locally at each party.

Definition 3 (Private Point Query)**.** *A private point query protocol between two mutually untrusting semi-honest parties,* Alice *and* Bob*, with inputs a point $pt \in \mathbb{R}^d$ and a d-polytope $\mathcal{P}$ respectively, returns to* Alice TRUE *iff $pt \in \mathcal{P}$ and* FALSE *otherwise.* Bob *learns nothing.*

The protocol only returns a boolean decision; when the protocol returns TRUE, Alice learns that pt lies inside or on the boundary of $\mathcal{P}$ and nothing more. If the protocol returns FALSE, alice does not learn any information about $\mathcal{P}$ including the relative positioning of pt to $\mathcal{P}$.

Definition 4 (Private Bounding Box Overlap Query)**.** *A private bounding box overlap query protocol between two mutually untrusting semi-honest parties with inputs cubioidal bounding boxes $\mathcal{B}_1$ and $\mathcal{B}_2$ respectively, returns to either of the parties* TRUE *iff $\mathcal{B}_1 \cap \mathcal{B}_2 \neq \emptyset$, and* FALSE *otherwise. The other party learns nothing.*

It is known that the task of computing the exact volume of the overlapping region of two polytopes with cost sub-exponential in the number of dimensions is intractable. Hence, we turn to an approximation which takes a *slack* parameter δ and an error probability ϵ, and computes an approximate volume of overlap [3].

1. To the best of our knowledge, proposals for inter-vehicle communication assume a global frame of reference (like GPS). A car's local view can be converted to this frame of reference using internal processing [31].

2. https://doc.cgal.org/latest/Convex_decomposition_3/index.html

$\mathcal{F}^d_{dot}$: Ideal Function for Vector Dot Product Range Check:

Parameters: Parties Alice and Bob. A Euclidean space $\mathbb{R}^d$, and a range of values $[\mathsf{lower}, \mathsf{upper}]$ where $\mathsf{lower}, \mathsf{upper} \in \mathbb{R}$, $\mathsf{upper} > \mathsf{lower}$.

Inputs: Alice has input $\vec{a} \in \mathbb{R}^d$ and Bob has input $\vec{b} \in \mathbb{R}^d$.

Output: Alice outputs TRUE if $\vec{a} \cdot \vec{b} \in [\mathsf{lower}, \mathsf{upper}]$ and FALSE otherwise. Bob outputs $\perp$.

$\mathcal{F}^d_{bmul}$: Ideal Function Blinded Matrix Multiplication:

Parameters: Parties Alice and Bob, and a finite field $\mathbb{F}$.

Inputs: Alice has input $\vec{a} \in \mathbb{F}^d$ and Bob has as inputs a matrix $F \in \mathbb{F}^{\phi \times d}$ and a randomly sampled vectors $\vec{r} \in \mathbb{F}^{\phi}$.

Output: Alice outputs $\vec{m}' := F \cdot \vec{a} + \vec{r}$ without learning F and $\vec{r}$. Bob outputs $\perp$.

Figure 3: Ideal functions for i) computing the dot product between two vectors and comparing the output against a range of values, and ii) computing a "blinded" dot product between a matrix and a vector.

Definition 5 (Private Volume of Overlap Estimation)**.** *A private approximate volume of overlap estimation protocol between two mutually untrusting semi-honest parties with inputs d-polytopes $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, returns to either of the parties $\mathsf{Vol}_{apprx} \in \mathbb{R}_{\geq 0}$ such that*

$$\mathbb{P}\left[\mathsf{Vol}_{int} - \delta \leq \mathsf{Vol}_{apprx} \leq \mathsf{Vol}_{int} + \delta\right] \geq 1 - \epsilon$$

where Vol_{int} is the volume of the overlapping region of $\mathcal{P}_1$ and $\mathcal{P}_2$, $\delta \in (0, 1)$ is an input slack parameter, and $\epsilon \in (0, 1)$ is an input error probability for the approximation.

5. Cryptographic Building Blocks

This section describes the tools that we will use to build our protocols in the next sections. As discussed before, we use simple and computationally-efficient tools keeping in mind that ultimately these protocols will be deployed on resource-constrained devices. As such, we will avoid expensive machinery, e.g., fully homomorphic encryption, despite the fact that in some cases they may come with additional benefits and features.

5.1. Privacy-Preserving Dot Product Computation & Range Check

In our protocols, we require a mechanism to check if the dot product between two vectors lies in a range of values, e.g., less than 0. The ideal function is defined in Fig. 3. There are several ways to realize this functionality; the obvious ones include using a garbled circuit or fully homomorphic encryption to compute the dot product and compare the output with the range end points. In addition, Atallah and Du [2] present a protocol that combines additively homomorphic encrytion with garbled circuits.

We take a more efficient alternative approach: we realize $\mathcal{F}^d_{dot}$ using a combination of oblivious linear evaluations (in a field), and a garbled circuit solving the millionaire's problem (or a private range checking protocol).

The idea is as follows (see Fig. 4): given $\vec{a} = \{a_1, \ldots, a_d\}$ and $\vec{b} = \{b_1, \ldots, b_d\}$, Alice and Bob map

Parameters: Parties Alice and Bob. Alice's input is a vector $\vec{a} = \{a_1, \ldots, a_d\} \in \mathbb{R}^d$, and Bob's input is a vector $\vec{b} = \{b_1, \ldots, b_d\} \in \mathbb{R}^d$. A finite field $\mathbb{F}$ of order $\mathrm{O}(\lambda)$ bits.

[Compute Blinded Result]

(1) Using a fixed point representation of $a_1, \ldots, a_d$, Alice maps the values to values in $\mathbb{F}$. Bob similarly maps the values of $b_1, \ldots, b_d$ to values in $\mathbb{F}$.
(2) Bob uniformly samples d elements $r_1, \ldots, r_d \xleftarrow{\$} \mathbb{F}$.
(3) For each $i \in [1, d]$, Alice and Bob call $\mathcal{F}_{ole}$:
- Alice's input is a_i
- Bob's input is b_i and r_i
- Alice learns $a_i b_i + r_i$.

(4) Alice computes $s = \sum_{i=1}^{d} a_i b_i + r_i$.

[Check Result]

To check if the dot product lies in some range [lower, upper], Bob generates a garbled circuit which takes as input s from Alice and $r = \sum_{i=1}^{d} r_i$. The circuit computes $s - r$ (in the field) and checks if the value lies in [lower, upper].

Figure 4: DotProdCheck: Private dot product of two vectors and checking if the output lies in a range of values.

their respective vector components to values in a finite field $\mathbb{F}$ using fixed-point representations. Then, Bob samples a vector $\vec{r} := [r_1, \ldots, r_d]$ with random elements from $\mathbb{F}$, and using d calls to $\mathcal{F}_{ole}$, Alice and Bob compute for each $i \in [1, d]$, $a_i b_i + r_i$ (see line 3 of Fig. 4). It may be evident that summing up these values provides the dot product of $\vec{a}$ and $\vec{b}$, blinded by the value of $\sum_{i=1}^{d} r_i$.

Subsequently, Alice and Bob "deblind" the values inside a garbled circuit generated by Bob and check if the values are in the provided range. This is a series of subtractions (in the field) followed by solving the millionaire's problem and is significantly faster than computing the dot product entirely inside the garbled circuit.

Mapping Inputs to Field Elements: Since the first part of our protocol computes on values in $\mathbb{F}$, we need to specifically define ranges to distinguish between positive and negative values in the garbled circuit later in the protocol. We use a fixed-point representation scheme that converts the coordinates of the points used in our protocol to integers in range $[-\rho, \rho]$, where ρ is fixed based on the size of the space and the desired level of precision. These integers are then mapped to field elements in $\mathbb{F}$.

Effectively, all vector components are mapped to field elements in $[-\rho, \rho]$. It is easy to see that in this case $\sum_{i=1}^{d} a_i b_i \in [|\mathbb{F}| - d\rho^2, d\rho^2]$ where the range $[0, d\rho^2]$ represents the case $\sum_{i=1}^{d} a_i b_i \geq 0$ and $[|\mathbb{F}| - d\rho^2, |\mathbb{F}| - 1]$ represents the case otherwise. To distinguish the two cases, we need to make sure that the ranges are disjoint; we require that $|\mathbb{F}| - d\rho^2 > d\rho^2$. As a running example, consider $d = 3$ and $\rho = 2^{32}$. Then, $|\mathbb{F}| \geq 2^{68}$ suffices.

Theorem 1. *Assuming that there is a protocol that realizes $\mathcal{F}_{ole}$ with $\mathrm{O}(\lambda)$ bits of communication,* DotProdCheck *realizes $\mathcal{F}_{dot}^{d}$ with $\mathrm{O}(d\lambda)$ bits communication.*

5.2. Blinded Matrix Dot Product Computation

Our protocols make extensive use of a primitive that computes a dot product of a vector input by one party with a set of vectors (matrix) input by the other party, and checks that outputs lie within a range of values. To achieve this, we will use a "blinded" dot product computation between a vector of one party and the rows of a matrix representing the vectors of the other party. The ideal function, $\mathcal{F}_{bmul}^{d}$, is defined in Fig. 3. This can be realized using $\mathcal{F}_{dot}^{d}$, i.e., computing the dot product between the input vector and each row of the matrix. However, a faster process exists using $\mathcal{F}_{vole}$ as the underlying primitive.

The idea is as follows: let $\vec{a}$ be the vector input by Alice and let F be the matrix input by Bob. In the protocol, Bob samples a matrix of $\phi \times d$ dimensions, R, where the elements of the ith row sum up to the ith component of $\vec{r}$ (see line 1 of Fig. 5). Then, using calls to $\mathcal{F}_{vole}$, Alice and Bob compute the matrix M' where the jth column of the matrix is computed as $a_j \times F[* : j] + R[* : j]$, i.e., a scalar product between the jth column of F and the jth component of $\vec{a}$ followed by an addition with the jth column of R (see line 2 of Fig. 5).

Summing up the values in the ith row of M' gives the dot product of $\vec{a}$ and the vector in the ith row of F blinded by the summation of the values in the ith row of R. Alice outputs the per-row sum of M' as the components of the output vector $\vec{m}'$ (see line 4 of Fig. 5). Depending on the protocol, the components of the blinded vector can then be checked either inside a garbled circuit, or with a dedicated private range checking algorithm [4].

Theorem 2. *Assuming that there is a protocol realizing $\mathcal{F}_{vole}$ for ϕ-length vectors with $\mathrm{O}(\phi\lambda)$ bits of communication,* BlindedMatrixMultiply *realizes $\mathcal{F}_{bmul}^{d}$ with $\mathrm{O}(\phi d\lambda)$ bits of communication.*

Parameters: Parties Alice and Bob and a finite field $\mathbb{F}$ of order $\mathrm{O}(\lambda)$ bits. Alice's input is a vector $\vec{a} = \{a_1, \ldots, a_d\} \in \mathbb{F}^d$, and Bob's inputs are: i) a matrix $F \in \mathbb{F}^{\phi \times d}$

$$F := \begin{bmatrix} c_{1,1} & \cdots & c_{1,d} \\ \vdots & & \vdots \\ c_{\phi,1} & \cdots & c_{\phi,d} \end{bmatrix}$$

and ii) a vector $\vec{r} \in \mathbb{F}^{\phi} = [r_1, \ldots, r_\phi]$ of random elements.

Protocol

(1) Bob samples a matrix $R \in \mathbb{F}^{\phi \times d}$

$$R := \begin{bmatrix} r_{1,1} & \ldots & r_{1,d} \\ \vdots & & \vdots \\ r_{\phi,1} & \ldots & r_{\phi,d} \end{bmatrix}$$

such that for $i \in [1, \phi]$, $\sum_{j=1}^{d} r_{i,j} = r_i$.
(2) Alice and Bob call $\mathcal{F}_{vole}$ d times such that in the j-th call:
- Alice's input is a_j.
- Bob's inputs are the vectors corresponding to the j-th column of F denoted by $F[* : j]$ and the j-th column of R denoted by $R[* : j]$.
- Alice learns a vector $[a_j c_{1,j} + r_{1,j}, \ldots, a_j c_{\phi,j} + r_{\phi,j}]$.

(3) Using the values obtained above, Alice computes the matrix

$$M' := \begin{bmatrix} a_1 c_{1,1} + r_{1,1} & \ldots & a_d c_{1,d} + r_{1,d} \\ \vdots & & \vdots \\ a_1 c_{\phi,1} + r_{\phi,1} & \ldots & a_d c_{\phi,d} + r_{\phi,d} \end{bmatrix}$$

(4) Alice computes and outputs the vector $\vec{m}' := [m'_1, \ldots, m'_\phi]$ such that $m'_i = \sum_{j=1}^{d} M'[i, j]$.

Figure 5: BlindedMatrixMultiply: Computation of dot product between a vector and a matrix blinded by random values.

Parameters: Parties Alice and Bob. Alice's input is a point $pt \in \mathbb{R}^d$ with position vector (with respect to the origin) $\vec{pt} = [x_1, \ldots, x_d]$. Bob's input is a closed, convex polytope $\mathcal{P}$ with ϕ faces in $\mathbb{R}^d$. A finite field $\mathbb{F}$ and a fixed-point representation of all coordinates that maps the positive coordinates (and 0) to $[0, \rho]$ and the negative coordinates to $[|\mathbb{F}|-\rho, |\mathbb{F}|-1]$ in $\mathbb{F}$.

[Compute Blinded Result Vector:]

(1) For the faces of $\mathcal{P}$, $f_1, \ldots, f_\phi$, Bob computes the inward facing normal vectors of the faces, $\vec{f_1}, \ldots, \vec{f_\phi}$ and correspondingly the face normal matrix

$$F = \begin{bmatrix} c_{1,1} & \cdots & c_{1,d} \\ \vdots & & \vdots \\ c_{\phi,1} & \cdots & c_{\phi,d} \end{bmatrix}$$

(2) Let $p_1, \ldots, p_\phi$ be a vertex on the faces $f_1, \ldots, f_\phi$ respectively. Bob computes the position vectors of $p_1, \ldots, p_\phi$, denoted $\vec{p_1}, \ldots, \vec{p_\phi}$, with respect to the origin, and computes $\vec{fp} = [x'_1, \ldots, x'_i]$ where $x'_i = \vec{f_i} \cdot \vec{p_i}$.

(3) Bob samples a random vector $\vec{r} = [r_1, \ldots, r_\phi] \in \mathbb{F}^\phi$.

(4) Alice and Bob call $\mathcal{F}^d_{bmul}$ with inputs $\vec{pt}$, and F, $\vec{r} - \vec{fp}$ respectively. This returns $\vec{m}'$ to Alice where for $i \in [1, \phi]$,

$$m'_i = \vec{pt} \cdot F[i : *] + r_i - x'_i$$

[Check Result Vector:]

(5) Bob generates a garbled circuit which takes $\vec{m}'$ as Alice's input and $\vec{r}$ as Bob's input. The circuit computes $\vec{m} := \vec{m}' - \vec{r}$, and checks that all the components are in the range $[0, d\rho^2]$.

(6) The circuit returns to Alice TRUE if the check returns true. Otherwise, the circuit returns FALSE. Alice outputs the result.

Figure 6: PointQueryGeneral: Protocol for privately determining if a point lies in an arbitrary d-polytope.

6. Private Point Query

In this section, we will present two protocols for private point queries (see Definition 3): a general protocol where the input is an arbitrary d-polytope (Sec. 6.1), and a more efficient protocol for d-simplices (Sec. 6.2).

6.1. Protocol for Arbitrary Polytopes

Intuition: The key idea behind the protocol is based on Fact 1. More specifically, let $pt \in \mathbb{R}^d$ be a secret point known to Alice while $\mathcal{P}$ is a convex polytope with ϕ faces, $f_1, \ldots, f_i$, known to Bob. For each face of $\mathcal{P}$, Bob computes the inward normal vector. Let $\{\vec{f_1}, \ldots, \vec{f_\phi}\}$ be the set of normal vectors. Also, let $p_1, \ldots, p_\phi$ be arbitrary points on $f_1, \ldots, f_\phi$ (say one of the vertices) respectively, and let $\vec{pt_i}$ is the position vector of pt *with respect to* p_i. Then, Alice and Bob jointly compute and check whether $\vec{pt_i} \cdot \vec{f_i} \geq 0$ for each $i \in [1, \phi]$.

First note that $\vec{pt_i} = \vec{pt} - \vec{p_i}$ where $\vec{pt}$ and $\vec{p_i}$ are position vectors with respect to the origin. Define $\vec{pt} = [x_1, \ldots, x_d]$ and let $\vec{f_i} = c_{i,1}\hat{x}_1 + \ldots + c_{i,d}\hat{x}_d$, for each $i \in [1, \phi]$. Finally, let $\vec{fp} = [x'_1, \ldots, x'_\phi]$ be a vector such that $x'_i = \vec{f_i} \cdot \vec{p_i}$. If $pt \in \mathcal{P}$, we have for each $i \in [1, \phi]$, $\vec{f_i} \cdot \vec{pt_i} = \vec{f_i} \cdot (\vec{pt} - \vec{p_i}) \geq 0$. In matrix form,

$$\overset{F}{\begin{bmatrix} c_{1,1} & \cdots & c_{1,d} \\ \vdots & & \vdots \\ c_{\phi,1} & \cdots & c_{\phi,d} \end{bmatrix}} \overset{\vec{pt}}{\begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}} - \overset{\vec{fp}}{\begin{bmatrix} x'_1 \\ \vdots \\ x'_\phi \end{bmatrix}} = \overset{\vec{m}}{\begin{bmatrix} m_1 \\ \vdots \\ m_\phi \end{bmatrix}} \geq \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

Our strategy to implement this check privately is a two-step process. Let $\vec{m} := F \cdot \vec{pt} - \vec{fp}$. In the first step, Alice learns $\vec{m}' := F \cdot \vec{pt} - \vec{fp} + \vec{r}$ where $\vec{r}$ is a vector with random elements sampled by Bob. In the second step, Alice and Bob use a generic 2PC circuit to check that for all $i \in [1, \phi], m'_i - r_i \geq 0$ where m'_i and r_i are the i-th component of $\vec{m}'$ and $\vec{r}$ respectively. This is essentially solving a millionaire's problem.

We will use $\mathcal{F}^d_{bmul}$ to compute $\vec{m}'$ in the first step (see lines 3–4 of Fig. 6). Then, the second step involves generating a garbled circuit distinguishing positive and negative values in the components of $\vec{m}'$ (see lines 5–6).

Communication Cost & Security: The interactive step of the protocol are the calls to $\mathcal{F}^d_{bmul}$ which requies $\mathrm{O}(\phi d\lambda)$ bits of communication and the garbled circuit computation, where Alice's and Bob's inputs are ϕ-length vectors. Thus, the overall communication cost of the protocol is $\mathrm{O}(\phi d\lambda)$ bits when BlindedMatrixMultiply is used to realize $\mathcal{F}^d_{bmul}$. The security of the scheme likewise depends on the security of BlindedMatrixMultiply securely realizing $\mathcal{F}^d_{bmul}$, which we show in Theorem 2.

Scaling to Multiple Polytopes: The protocol described in Fig. 6 considers that Alice's input is a point pt while Bob's input is a polytope $\mathcal{P}$. The protocol can be easily adapted to the scenario where Bob's input is a set of polytopes, $\mathcal{P}_1, \ldots, \mathcal{P}_n$ with ϕ faces each. Instead of running n instances of PointQueryGeneral with pt and each of $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ as input, we leverage the fact that $\mathcal{F}_{vole}$ can be used to batch the dot-product computation across all instances with low amortized communication and compute costs. This leads to an optimized protocol that scales better with n (as we will demonstrate in Sec. 9).

Specifically, consider that $F_1, \ldots, F_n$ are the face normal matrices for these polytopes. Consider the matrix $F' = F_1 || \ldots || F_n \in \mathbb{R}^{n\phi \times d}$ where F' is generated by concatenating $F_1, \ldots, F_n$ row-wise. Then, $(pt \in \mathcal{P}_1) \vee \cdots \vee (pt \in \mathcal{P}_n)$ iff there is some $t \in [1, n]$ such that $F_t \cdot \vec{pt} - \vec{fp}_t \geq [0, \ldots, 0]$. For this the protocol computes a "blinded" version of

$$\overset{F'}{\begin{bmatrix} c_{1,1} & \cdots & c_{1,d} \\ \vdots & & \vdots \\ c_{n\phi,1} & \cdots & c_{n\phi,d} \end{bmatrix}} \overset{\vec{pt}}{\begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}} - \overset{\vec{fp}'}{\begin{bmatrix} x'_1 \\ \vdots \\ x'_{n\phi} \end{bmatrix}}$$

This computation is performed using d calls to $\mathcal{F}_{vole}$ with Bob's inputs being columns of F' and random vectors of length $n\phi$. The rest of the steps of the protocol require only minor modifications to account for the fact that there are n polytopes. We omit further details.

6.2. Protocol for Simplex

Intuition: Our protocol expands on the discussion in Sec. 3. Utilizing the properties of a simplex, we are able to replace a garbled circuit with a dedicated private range checking protocol, and eliminate several expensive steps including pre-computing the faces and the face normal vector. Specifically, given a d-simplex with the set of vertices $\{v_1, \ldots, v_{d+1}\}$ and a point $pt = (x_1, \ldots, x_d) \in \mathbb{R}^d$, let $(a_1, \ldots, a_{d+1})$ be the barycentric coordinates of pt relative to the simplex. Then we have for each $i \in [1, d+1]$, $a_i \in [0, 1]$ iff pt lies inside (or on the boundary) of the

Parameters: Parties Alice and Bob. Alice's input is a point $pt \in \mathbb{R}^d$ with position vector $\vec{pt} = [x_1, \ldots, x_d]$. Bob's input is a d-simplex in $\mathbb{R}^d$ with $\{v_1, \ldots, v_{d+1}\}$ being the vertices of the simplex. A finite field $\mathbb{F}$ and a fixed-point representation of all coordinates that maps the positive coordinates (and 0) to $[0, \rho]$ and the negative coordinates to $[|\mathbb{F}| - \rho, |\mathbb{F}| - 1]$ in $\mathbb{F}$. Precision γ for the fixed-point representation.

[Compute Blinded Coordinates:]
(1) Bob samples a random vector $\vec{r} = [r_1, \ldots, r_d] \in \mathbb{F}^d$,
(2) Bob computes V and its inverse where

$$V := \begin{bmatrix} v_{1,1} - v_{d+1,1} & \ldots & v_{1,d} - v_{d+1,d} \\ \vdots & & \vdots \\ v_{d,1} - v_{d+1,1} & \ldots & v_{d,d} - v_{d+1,d} \end{bmatrix}$$

$$V^{-1} := \begin{bmatrix} v'_{1,1} & \ldots & v'_{1,d} \\ \vdots & & \vdots \\ v'_{d,1} & \ldots & v'_{d,d} \end{bmatrix}$$

(3) Alice and Bob call $\mathcal{F}^d_{bmul}$ where Alice's input is $\vec{pt}$, and Bob's inputs are V^{-1} and the vector $\vec{r} - V^{-1} \cdot v_{\vec{d+1}}$.
(4) Alice receives $\vec{a}'$ from $\mathcal{F}^d_{bmul}$. Alice computes $a'_{d+1} = 1 \times 10^\gamma - \sum_{i=1}^{d} \vec{a}'[i]$.

[Check Barycentric Coordinates:]
(5) Bob generates a garbled circuit where Alice's input is $\vec{a}'$ and Bob's input is $\vec{r}$. The circuit computes $\vec{m} := \vec{a}' - \vec{r}'$, and checks that all the components of the vectors are in $[0, 10^\gamma]$.
(6) The circuit returns to Alice TRUE if the check returns TRUE. Otherwise, the circuit returns FALSE. Alice outputs the result.

Figure 7: PointQuerySimplex: Protocol for privately determining if a point lies in a d-simplex.

simplex. Mathematically, let $v_i = (v_{i,1}, \ldots, v_{i,d}) \in \mathbb{R}^d$. The barycentric coordinates of pt are as follows.

$$\underbrace{\begin{bmatrix} v_{1,1} - v_{d+1,1} & \ldots & v_{1,d} - v_{d+1,d} \\ \vdots & & \vdots \\ v_{d,1} - v_{d+1,1} & \ldots & v_{d,d} - v_{d+1,d} \end{bmatrix}}_{V} \underbrace{\begin{bmatrix} a_1 \\ \vdots \\ a_d \end{bmatrix}}_{\vec{a}} = \underbrace{\begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}}_{\vec{pt}} - \underbrace{\begin{bmatrix} v_{d+1,1} \\ \vdots \\ v_{d+1,d} \end{bmatrix}}_{\vec{v}_{d+1}}$$

In our protocol, we will compute $\vec{a}$

$$\vec{a} = V^{-1} \cdot (\vec{pt} - \vec{v}_{d+1})$$

where $\vec{pt}$ and $\vec{v}_{d+1}$ are the position vectors of pt and v_{d+1}. In order to compute $V^{-1} \cdot (\vec{pt} - \vec{v}_{d+1})$, the protocol (see Fig. 7) uses $\mathcal{F}^d_{bmul}$. Alice's input to $\mathcal{F}^d_{bmul}$ is $\vec{pt}$ and Bob's inputs are V^{-1} and the vector $\vec{r} - V^{-1} \cdot \vec{v}_{d+1}$ where $\vec{r} \in \mathbb{F}^d$ is a vector of random elements (see line 3 of Fig. 7). $\mathcal{F}^d_{bmul}$ returns to Alice $\vec{a}$ "blinded" by $\vec{r}$. These values are then "de-blinded", and the components are checked against the range of values in $[0, 1]$ (adjusted to the fixed-point representation) (see line 5 of Fig. 7). For simplicity, the protocol description in Fig. 7 shows how to realize this using a garbled circuit, and we will desribe later how this can be replaced with a dedicated private range checking protocol. Also, the protocol checks that $a_{d+1} = 1 - \sum_{i=1}^{d} a_i \in [0, 1]$ (see line 4 of Fig. 7).

Communication Cost & Security: The communication cost of the protocol is straightforward: the interactive step calls $\mathcal{F}^d_{bmul}$ with the a matrix of dimension $d \times d$ and a vector of length d. The overall communication complexity is $\mathrm{O}(d^2\lambda)$ bits of communication when using BlindedMatrixMultiply as the protocol realizing $\mathcal{F}^d_{bmul}$. The security of the scheme also relies on the security of BlindedMatrixMultiply, which we show in Theorem 2.

Parameters: Parties Alice and Bob. Inputs $\mathcal{B}_{\mathsf{Alice}}$ and $\mathcal{B}_{\mathsf{Bob}}$. Let $pt_a = (pt_{a,1}, \ldots, pt_{a,d})$ and $pt_b = (pt_{b,1}, \ldots, pt_{b,d})$ be the coordinates of the topmost, left corner of $\mathcal{B}_{\mathsf{Alice}}$ and $\mathcal{B}_{\mathsf{Bob}}$ respectively. Let $\ell_{a,1}, \ldots, \ell_{a,d}$ and $\ell_{b,1}, \ldots, \ell_{b,d}$ be the lengths of the edges of $\mathcal{B}_{\mathsf{Alice}}$ and $\mathcal{B}_{\mathsf{Bob}}$ respectively along the d axes.

[Detect Bounding Box Overlap]
(1) Bob creates a garbled circuit which computes $\mathcal{B}_{\mathsf{Alice}} \ominus \mathcal{B}_{\mathsf{Bob}}$ as follows:
a) Compute $pt_r = (pt_{r,1}, \ldots, pt_{r,d}) \in \mathbb{R}^d$ such that $pt_{r,i} := pt_{a,i} - pt_{b,i} - \ell_{b,i}$.
b) For $m \in [1, d]$, compute $\ell_{r,m} = \ell_{a,m} + \ell_{b,m}$.
c) For $m \in [1, d]$, check if $pt_{r,m} \leq 0 \leq pt_{r,m} + \ell_{r,m}$.
d) Return TRUE iff all the the above checks return TRUE.

Figure 8: OverlapAABB: Protocol to detect if two d-dimensional axis-aligned bounding boxes overlap.

Replacing the Garbled Circuit with Range Checking: An advantage of barycentric coordinates is that since the components of $\vec{a}$ have to be checked against a range of values with defined upper and lower limits, we can employ a dedicated private range checking algorithm [4] instead of a garbled circuit to solve the millionaire's problem. Casting this to our protocol, we will check that all components of $\vec{a}'$ are in the range $[r, r + 10^\gamma]$ (in the field) where γ is the precision parameter of the fixed point representation. The communication cost to check each component in this case scales linearly in γ. More details can be found in [4].

7. Private Bounding Box Overlap Query

This section provides protocols for private bounding box overlap queries (see Definition 4) . We consider two types of bounding boxes: i) axis-aligned boxes (AABB), and ii) oriented boxes (OBB). AABBs are rectangular bounding boxes where the edges of the box are aligned with the canonical axes of the space in which the box resides, and are used in majority of the applications since they are easy to compute and detect overlaps with. OBBs are rectangular bounding boxes where the edges are not aligned with the axes of the space in which the box resides and may often better describe a complex shape. First, we will describe a simple protocol in Sec. 7.1 to detect overlaps of axis-aligned boxes. Then, we will provide a protocol detecting overlaps of oriented boxes in Sec. 7.2.

7.1. Detecting Overlap of Axis-aligned Boxes

We can detect when two axis-aligned bounding boxes overlap by checking if any of the vertices of one box lies within the other box. There are 2^d vertices and a protocol built on this idea will have costs scaling exponentially in d. A more optimized protocol with cost scaling linearly in d can be constructed based on Fact 3. Let $\mathcal{B}_{\mathsf{Alice}}$ and $\mathcal{B}_{\mathsf{Bob}}$ be the bounding box inputs of Alice and Bob. Alice and Bob compute the Minkowski difference of $\mathcal{B}_{\mathsf{Alice}}$ and $\mathcal{B}_{\mathsf{Bob}}$ and then check if the origin is in $\mathcal{B}_{\mathsf{Alice}} \ominus \mathcal{B}_{\mathsf{Bob}}$.

The Minkowski difference of two axis-aligned boxes is also an axis-aligned box and is easy to compute [22]. Specifically, let $\mathcal{B}_{\mathsf{Alice}}$ and $\mathcal{B}_{\mathsf{Bob}}$ be two axis-aligned bounding boxes with $pt_a = (pt_{a,1}, \ldots, pt_{a,d}) \in \mathbb{R}^d$ and $pt_b = (pt_{b,1}, \ldots, pt_{b,d}) \in \mathbb{R}^d$ being the coordinates of the top, leftmost corner vertex of $\mathcal{B}_{\mathsf{Alice}}$ and $\mathcal{B}_{\mathsf{Bob}}$ respectively. Also, let the lengths of the edges of $\mathcal{B}_{\mathsf{Alice}}$ along the d-axes be $\ell_{a,1}, \ldots, \ell_{a,d}$. Similarly, let the lengths

Parameters: Parties Alice and Bob. Inputs $\mathcal{B}_{\mathsf{Alice}}$ and $\mathcal{B}_{\mathsf{Bob}}$ that have arbitrary orientations. Let the set of edges of $\mathcal{B}_{\mathsf{Alice}}$ be $\{\mathsf{Edge}_1, \ldots, \mathsf{Edge}_t\}$ where $t = 2^d$ and the set of faces of $\mathcal{B}_{\mathsf{Bob}}$ be $\{\mathsf{Face}_1, \ldots, \mathsf{Face}_\phi\}$. A finite field $\mathbb{F}$ and a fixed-point representation of all coordinates that maps the positive coordinates (and 0) to $[0, \rho]$ and the negative coordinates to $[|\mathbb{F}| - \rho, |\mathbb{F}| - 1]$ in $\mathbb{F}$.

[Detect Edge-Face Intersection]

(1) Let the equation of the plane corresponding to Face_j be $c_{j,1}x_1 + \cdots + c_{j,d}x_d + c_{j,d+1} = 0$. Bob computes the matrix

$$F := \begin{bmatrix} c_{1,1} & \ldots & c_{1,d} \\ \vdots & & \vdots \\ c_{\phi,1} & \ldots & c_{\phi,d} \end{bmatrix}$$

(2) Bob samples two random vectors $\vec{r}_1, \vec{r}_2 \in \mathbb{F}^\phi$.

(3) Alice and Bob repeat the following for each edge $\{\mathsf{Edge}_1, \ldots, \mathsf{Edge}_t\}$

a) Let $pt_1 := (x_{1,1}, \ldots, x_{1,d})$ and $pt_2 := (x_{2,1}, \ldots, x_{2,d})$ be the end points of the line segment representing the edge, and let $\vec{pt_1}$ and $\vec{pt_2}$ be their position vectors respectively.

b) Alice and Bob call $\mathcal{F}^d_{bmul}$ twice: in the first call, Alice's input is $\vec{pt_1}$, and Bob's inputs are F and $\vec{r}_1$, while in the second call Alice's input is $\vec{pt_2}$, and Bob's inputs are F and $\vec{r}_2$. Alice's outputs are $\vec{m}'_1$ and $\vec{m}'_2$ respectively.

c) Using a garbled circuit generated by Bob, Alice and Bob compute $\vec{m}_1 := \vec{m}'_1 - \vec{r}_1$ and $\vec{m}_2 := \vec{m}'_2 - \vec{r}_2$. Let $\vec{m}_1 = [m_{1,1}, \ldots, m_{1,\phi}]$, $\vec{m}_2 = [m_{2,1}, \ldots, m_{2,\phi}]$. The circuit checks if there is some $i \in [1, \phi]$, $\mathbf{sgn}(m_{1,i}) \neq \mathbf{sgn}(m_{2,i})$, i.e.,

$$\begin{array}{rl} & \big((m_{1,i} \in [0, d\rho^2]) \wedge (m_{2,i} \in [|\mathbb{F}| - d\rho^2, 0])\big) \\ \vee & \big((m_{1,i} \in [|\mathbb{F}| - d\rho^2, 0]) \wedge (m_{2,i} \in [0, d\rho^2])\big) \end{array} \quad (1)$$

If so, then output TRUE.

[Detect Enclosing Object]

Alice selects an arbitrary point pt_a from within $\mathcal{B}_{\mathsf{Alice}}$. Alice and Bob call $\mathcal{F}^d_{ppq}$ with pt_a and $\mathcal{B}_{\mathsf{Bob}}$ as inputs. If $\mathcal{F}^d_{ppq}$ outputs TRUE, then output TRUE. Bob selects an arbitrary point pt_b from within $\mathcal{B}_{\mathsf{Bob}}$. Alice and Bob call $\mathcal{F}^d_{ppq}$ with pt_b and $\mathcal{B}_{\mathsf{Alice}}$ as inputs. If $\mathcal{F}^d_{ppq}$ outputs TRUE, then output TRUE.

Figure 9: OverlapOBB: Protocol for detecting if two d-dimensional arbitrarily oriented bounding boxes overlap.

of the edges of $\mathcal{B}_{\mathsf{Bob}}$ along the axes be $\ell_{b,1}, \ldots, \ell_{b,d}$. If $pt_r = (pt_{r,1}, \ldots, pt_{r,d})$ is the top, leftmost corner vertex of $\mathcal{B}_{\mathsf{Alice}} \ominus \mathcal{B}_{\mathsf{Bob}}$, then for $m \in [1, d]$

$$pt_{r,m} = pt_{a,m} - pt_{b,m} - \ell_{b,m}$$

Similarly, the length of the edges of $\mathcal{B}_{\mathsf{Alice}} \ominus \mathcal{B}_{\mathsf{Bob}}$ along the d axes, $m \in [1, d]$ is given by

$$\ell_{r,m} := \ell_{a,m} + \ell_{b,m}$$

OverlapAABB (Fig. 8) realizes this idea using a garbled circuit, which computes $\mathcal{B}_{\mathsf{Alice}} \ominus \mathcal{B}_{\mathsf{Bob}}$ (lines 1a–1b) and then checks if the origin lies inside (line 1c)

Communication Cost & Security: OverlapAABB requires $\mathrm{O}(d)$ bits of communication as the lengths of the edges along each axis is input into the garbled circuit by either party. The security of the scheme is straightforward: any scheme implementing Yao's garbled circuit for two-party can be used to securely implement OverlapAABB.

7.2. Detecting Overlap of Oriented Boxes

Detecting if two oriented boxes overlap is significantly more complex than the case for axis-aligned boxes [3]. In a non-private setting, the simplest way to detect if two oriented boxes intersect is by using the separating axis theorem (SAT) whereby the protocol tries to find a axis such that the projections of the boxes along this axis are disjoint (see Sec. 3.1).

In order to implement SAT over two boxes privately, the axes along which the projections are checked are also generated securely. That is, the parties input the vector corresponding to the edges of their boxes into a secure computation framework (e.g., a garbled circuit) which i) computes the axes by computing the cross product of the edges, ii) computes the projections of the two boxes on the axes, and iii) checks if there is an axis along which the ranges corresponding to the projections are disjoint. Due to the vector operations inside the garbled circuit, the process is computationally expensive.

We will present a computationally faster (albeit less sophisticated) protocol that can be realized in a private setting with lower communication and compute overheads for low dimensions. The protocol is based on the following

Fact 4 ([17])**.** *When two cuboidal boxes overlap, one of two cases are possible: i) there is at least one edge of one box that intersects with one (or more) face of the other box, or ii) one box completely encloses the other box.*

In our protocol, we will check for both conditions. When the second condition holds, any arbitrary point selected from the inner box will lie inside the outer box. We will employ a private point query protocol to check if a point in one box lies inside the other box. The first condition requires more attention and for this we will rely on the fact that we can detect when a line segment intersects a plane by simply plugging in the coordinates of the end-points of the line segment in the equation of the plane and check if the values obtained are of opposite signs. Specifically, consider that the point $pt_1 = (x_{1,1}, \ldots, x_{1,d})$ and $pt_2 = (x_{2,1}, \ldots, x_{2,d})$ are the end points of a line segment, and $c_1x_1 + \cdots + c_dx_d + c_{d+1} = 0$ is the equation of a plane. Then, the line segment intersects the plane iff

$$\mathbf{sgn}(c_1x_{1,1} + \cdots + c_{d+1}) \neq \mathbf{sgn}(c_1x_{2,1} + \cdots + c_{d+1}) \quad (2)$$

OverlapOBB (Fig. 9) describes a private protocol based on this idea. Bob first computes the matrix F using the equations of all of the ϕ faces of $\mathcal{B}_{\mathsf{Bob}}$ (line 1).

$$F := \begin{bmatrix} c_{1,1} & \ldots & c_{1,d} \\ \vdots & & \vdots \\ c_{\phi,1} & \ldots & c_{\phi,d} \end{bmatrix}$$

Then, for the end-points of each edge in $\mathcal{B}_{\mathsf{Alice}}$, pt_1 and pt_2 and their position vectors $\vec{pt_1}$ and $\vec{pt_2}$, Alice and Bob call $\mathcal{F}^d_{bmul}$ twice (see line 4b of Fig. 9): in the first call Alice's input is $\vec{pt_1}$ and Bob's inputs are F and a randomly sampled vector $\vec{r}_1 \in \mathbb{F}^\phi$, and in the second call Alice's input is $\vec{pt_2}$ and Bob's inputs are F and a randomly sampled vector $\vec{r}_2 \in \mathbb{F}^\phi$. Alice learns the dot product of $\vec{pt_1}$ and each of the face normal vectors in F blinded by $\vec{r}_1$ in $\vec{m}'_1$. Similarly, Alice learns $\vec{m}'_2$ respectively. With these as Alice's input and $\vec{r}_1$ and $\vec{r}_2$ as Bob's inputs, Alice and Bob "deblind" the values inside a garbled circuit and check if (2) holds (see line 4c).

3. The idea of using the Minkowski difference can be applied to oriented boxes as well, however computing the Minkowski difference of two oriented boxes is expensive inside a secure computation framework

Parameters: Parties Alice and Bob. Alice's input is a closed convex body $\mathcal{C}_{\mathsf{Alice}}$ and Bob's input is a closed convex body $\mathcal{C}_{\mathsf{Bob}}$ in $\mathbb{R}^d$. A jointly selected slack parameter, $\delta \in (0,1)$.

[Compute Approximate Volume of $\mathcal{C}_{\mathsf{Alice}} \cap \mathcal{C}_{\mathsf{Bob}}$]

(1) Using VolumeQuery, Alice and Bob determine the volumes of $\mathcal{C}_{\mathsf{Alice}}$ and $\mathcal{C}_{\mathsf{Bob}}$: $\mathsf{Vol}_{\mathsf{Alice}} := \mathsf{Vol}(\mathcal{C}_{\mathsf{Alice}})$, $\mathsf{Vol}_{\mathsf{Bob}} := \mathsf{Vol}(\mathcal{C}_{\mathsf{Bob}})$.

(2) Assuming $\mathsf{Vol}(\mathcal{C}_{\mathsf{Alice}}) < \mathsf{Vol}(\mathcal{C}_{\mathsf{Bob}})$ let $\mathsf{Vol}_{\mathsf{min}} := \mathsf{Vol}_{\mathsf{Alice}}$;

(3) Alice randomly samples $N = \mathrm{O}\left(\frac{1}{\delta^2}\right)$ point $pt_1, \ldots, pt_N$ such that $pt_k := \mathsf{SampleQuery}(\mathcal{C}_{\mathsf{Alice}})$ where $k \in [1, N]$

(4) Alice initializes a set of indicator variables $\{Z_1, \ldots, Z_N\}$. For each $k \in [1, N]$, Alice calls $\mathsf{PointQuery}(pt_k, \mathcal{C}_{\mathsf{Bob}})$. If PointQuery returns TRUE for pt_k, Alice sets $Z_k = 1$. Otherwise, Alice sets $Z_k = 0$.

(5) Alice outputs $\mathsf{Vol}_{apprx} := \mathsf{Vol}_{\mathsf{min}} \times \frac{1}{N}(Z_1 + \cdots + Z_N)$.

Figure 10: Non-private approximation of volume of overlapping regions [3]

Communication Cost & Security: OverlapOBB makes $2\times$ the number of edges calls to $\mathcal{F}^d_{bmul}$, each call processing a specific vertex of Alice's input box. Implemented naively (as in Fig. 9), this would process each vertex multiple times since a vertex is shared by multiple edges. Alternatively, it suffices to process each vertex separately, and then use the results to check (2). In this way, the number of calls to $\mathcal{F}^d_{bmul}$ is equal to the number of the vertices of the box, which is 2^d. The overall communication complexity is $\mathrm{O}\left(2^d \phi d \lambda\right)$ bits of communication when BlindedMatrixMultiply is used to implement $\mathcal{F}^d_{bmul}$. The security of the protocol relies on BlindedMatrixMultiply securely realizing $\mathcal{F}^d_{bmul}$ (shown in Theorem 2), and a secure implementation of a garbled circuit.

Scaling to Multiple Boxes: OverlapOBB assumes that Alice's input is a bounding box $\mathcal{B}_{\mathsf{Alice}}$ and Bob's input is $\mathcal{B}_{\mathsf{Bob}}$. The protocol can be adapted to the case where Bob's input is a set of n boxes. Instead of running n instances of OverlapOBB with each of Bob's input we can use the optimization proposed for PointQueryGeneral (see Sec. 6) by batching the dot-product computations required across all boxes (see line 4b of Fig. 9 for reference) using $\mathcal{F}_{vole}$. The resulting protocol has lower amortized communication cost and compute time. We omit further details as the steps are similar to the ones in Sec. 6.

8. Measuring Volume of Overlap

Even in a non-private setting, computing the intersection of simple closed convex bodies, e.g., axis-aligned boxes, given the vertices of the boxes is challenging. It is known that the problem is #P-hard, and there is no algorithm with costs scaling polynomially in the number of dimensions. To circumvent this problem, there is a line of work on approximating the volume of intersection for a large class of bodies. The algorithm (see Algorithm 10) relies on three types of *oracles* for answering queries:

(1) **Point Query Oracle** $\mathsf{PointQuery}(pt, \mathcal{C})$: Given a point $pt \in \mathbb{R}^d$ and a convex body $\mathcal{C}$, it return TRUE iff there is a body $\mathcal{C}'$ in $\mathbb{R}^d$ such that $pt \in \mathcal{C}'$ *and* $\mathsf{Vol}((\mathcal{C}' \setminus \mathcal{C}) \cup (\mathcal{C} \setminus \mathcal{C}')) \leq \epsilon_p \mathsf{Vol}(\mathcal{C})$ where $\epsilon_p \in (0,1)$.

(2) **Volume Query Oracle** $\mathsf{VolumeQuery}(\mathcal{C})$: Returns a value Vol_{apprx} such that $(1-\epsilon_v)\ \mathsf{Vol}(\mathcal{C}) \leq \mathsf{Vol}_{apprx} \leq (1-\epsilon_v)\ \mathsf{Vol}(\mathcal{C})$ where $\epsilon_v \in (0,1)$.

Parameters: Parties Alice and Bob. Alice's input is a d-polytope $\mathcal{P}_{\mathsf{Alice}}$ and Bob's input is a d-polytope $\mathcal{P}_{\mathsf{Bob}}$. Alice and Bob have access to volume and sample query oracles, VolumeQuery and SampleQuery respectively. Alice and Bob fix an error probability $\epsilon \in (0,1)$ and a slack parameter $\delta \in (0,1)$.

[Compute Approximate Volume of $\mathcal{P}_{\mathsf{Alice}} \cap \mathcal{P}_{\mathsf{Bob}}$]

(1) Alice and Bob compute $\mathsf{Vol}_{\mathsf{Alice}} := \mathsf{VolumeQuery}(\mathcal{P}_{\mathsf{Alice}})$ and $\mathsf{Vol}_{\mathsf{Bob}} := \mathsf{VolumeQuery}(\mathcal{P}_{\mathsf{Bob}})$. Alice and Bob privately determine $\min(\mathsf{Vol}_{\mathsf{Alice}}, \mathsf{Vol}_{\mathsf{Bob}})$ using a garbled circuit solving the millionaire's problem taking $\mathsf{Vol}_{\mathsf{Alice}}$ and $\mathsf{Vol}_{\mathsf{Bob}}$ as inputs. W.l.o.g, let this be $\mathcal{P}_{\mathsf{Alice}}$.

(2) Alice and Bob perform the following steps for $\tau = \frac{4}{3} \times \ln(\frac{1}{\epsilon})$ rounds. In the r-th round

a) Alice samples $N = \mathrm{O}\left(\frac{1}{\delta^2}\right)$ points from within $\mathcal{P}_{\mathsf{Alice}}$, $\{pt_{r,1}, \ldots, pt_{r,N}\}$ using $\mathsf{SampleQuery}(\mathcal{P}_{\mathsf{Alice}})$.

b) Alice initializes a set of indicator variables $\{Z_{r,1}, \ldots, Z_{r,N}\}$. For $k \in [1, N]$, Alice and Bob call PointQueryGeneral with $pt_{r,k}$ and $\mathcal{C}_{\mathsf{Bob}}$ as input. If PointQueryGeneral returns TRUE for $pt_{r,k}$, Alice sets $Z_{r,k} = 1$. Otherwise, $Z_{r,k} = 0$.

c) Alice computes $\mathsf{Vol}_r := \mathsf{Vol}_{\mathsf{Alice}} \times \frac{1}{N}(Z_{r,1} + \cdots + Z_{r,N})$

(3) Alice outputs Vol_{apprx} as the median of the values $\mathsf{Vol}_1, \ldots, \mathsf{Vol}_\tau$ obtained in the τ rounds above.

Figure 11: ApprxOverlapVolume: Protocol for privately approximating the volume of intersection of two d-polytopes.

(3) **Sample Query Oracle** $\mathsf{SampleQuery}(\mathcal{C})$: Loosely put, the oracle returns a uniformly random point sampled from within $\mathcal{C}$. [4]

The algorithm outputs a value $\mathsf{Vol}_{apprx} \in \mathbb{R}_{\geq 0}$ and Bringmann and Friedrich [3] show that

$$\mathbb{P}\left[\mathsf{Vol}_{int} - \delta \cdot \mathsf{Vol}_{\mathsf{min}} \leq \mathsf{Vol}_{apprx} \leq \mathsf{Vol}_{int} + \delta \cdot \mathsf{Vol}_{\mathsf{min}}\right] \geq \tfrac{3}{4}$$

where $\mathsf{Vol}_{int} := \mathsf{Vol}(\mathcal{C}_{\mathsf{Alice}} \cap \mathcal{C}_{\mathsf{Bob}})$. The error probability $c = 1 - \frac{3}{4}$ can be diminished through *probability amplification* techniques[5]. Specifically, for a success probability of $1-\epsilon$, the protocol is repeated $\mathbf{poly}(\epsilon, c)$ times.

8.1. Private Approxinate Volume Estimation

We will now present a private protocol for approximating the volume of the overlapping region of two d-polytopes satisfying Definition 5. The protocol denoted ApprxOverlapVolume, implements Algorithm 10 using a private point query protocol instantiated either with PointQueryGeneral or PointQuerySimplex depending on the type of input. Alice and Bob select an error probability $\epsilon \in (0,1)$ and a slack parameter $\delta \in (0,1)$. They determine $\mathsf{Vol}_{\mathsf{Alice}} = \mathsf{VolumeQuery}(\mathcal{P}_{\mathsf{Alice}})$ and $\mathsf{Vol}_{\mathsf{Bob}} = \mathsf{VolumeQuery}(\mathcal{P}_{\mathsf{Bob}})$ and $\mathsf{min}(\mathsf{Vol}_{\mathsf{Alice}}, \mathsf{Vol}_{\mathsf{Bob}})$ (see line 1 of Fig. 11). Assuming w.l.o.g. that $\mathsf{min}(\mathsf{Vol}_{\mathsf{Alice}}, \mathsf{Vol}_{\mathsf{Bob}}) = \mathsf{Vol}_{\mathsf{Alice}}$, the rest of the steps are initiated by Alice.

Alice repeats the following steps for $\tau := \mathrm{O}\left(\ln \frac{1}{\epsilon}\right)$ rounds: Alice uniformly samples $N = \mathrm{O}\left(\frac{1}{\delta^2}\right)$ uniformly random points $pt_1, \ldots, pt_N$ from within $\mathcal{P}_{\mathsf{Alice}}$ using $\mathsf{SampleQuery}(\mathcal{P}_{\mathsf{Alice}})$. Alice and Bob call a private point query protocol N times, with pt_k as Alice's input in the k-th round, and $\mathcal{P}_{\mathsf{Bob}}$ as Bob's input (line 2). For each point that the query returns TRUE, Alice increments a counter. The approximate volume of overlap of $\mathcal{P}_{\mathsf{Alice}}$ and $\mathcal{P}_{\mathsf{Bob}}$ is proportional to the total value of the counter after executing point queries over all N points. The protocol

4. Bringmann and Friedrich [3] formalize this idea to achieve *almost* uniform random sampling for a body in $\mathbb{R}^d$. We refer to that paper for more details on how to realize this oracle.

5. https://www.cs.huji.ac.il/course/2006/tcsg/Tirgul/tirgul1.pdf

outputs the median value of all the values computed in the previous τ rounds (line 3).

Volume of Overlap for Axis-Aligned Boxes: When applying the protocol to the special case of axis-aligned bounding boxes, we can apply PointQueryGeneral or PointQuerySimplex to determine if a point lies inside a box. However, a simpler and more efficient mechanism exists to implement this check. Specifically, consider a point $pt = (x_1, \ldots x_d)$ and an axis-aligned bounding box $\mathcal{B}$ where each point inside (or on the boundary) of the box lies within the range of values $[x'_{m,0}, x'_{m,1}]$ along the m-th axis. This range of values for each axis can be easily determined from the vertices of the box. Then, $pt \in \mathcal{B}$ iff

$$(x_1 \in [x'_{1,0}, x'_{1,1}]) \wedge \cdots \wedge (x_d \in [x'_{d,0}, x'_{d,1}])$$

The check can be implemented by solving d instances of the millionaire's problem and a series of logical AND operations in a garbled circuit; we omit the details of this straightforward implementation.

Communication Cost & Security: The communication cost of the protcol depends on the instantiation of private point queries. When instantiated with PointQueryGeneral, ApprxOverlapVolume requires $\mathrm{O}\left(\frac{1}{\delta^2} \ln \frac{1}{\epsilon} \cdot \phi d \lambda\right)$ bits of communication where the ϕ is the number of faces of Bob's input polytope. Also, the security of the protocol directly reduces to the security of the protocol used to realize private point queries. For the protocols presented in this paper, the security of ApprxOverlapVolume reduces to the arguments presented in Sec. 6 showing security of PointQueryGeneral and PointQuerySimplex.

9. Evaluation

We implemented all the protocols presented in this paper in C++11. The implementations use the NTL library [37] for implementing the finite field arithmetic, and the operations are performed over a 128-bit prime-order field. We used the open-source implementation[6] of the state-of-the-art VOLE scheme [42]. We implemented garbled circuits with the *emp-toolkit* open-source library[7]. The protocol of Atallah and Du [2] was implemented with Paillier encryption scheme with 2048-bits keys.

Datasets: We used a combination of real-world datasets and randomly generated datasets for benchmarking.

(1) **Randomly-generated Polytopes:** We created a dataset with 100 randomly generated polytopes with different specifications using Polymake [15]. To generate an arbitrary polytope in $\mathbb{R}^2$ and $\mathbb{R}^3$, randomly sampled points are provided as vertices of the polytope. In 2D space, the polygons generated have 3–16 vertices, and in 3D space the polytopes generated have 4–18 vertices; note that the simplest polygon is a triangle with 3 vertices and the simplest polytope in 3D is a tetrahedron with 4 vertices.

(2) **Axis-aligned bounding boxes from CARLA:** We used the open-source CARLA autonomous driving simulator [8] to generate a dataset of bounding boxes of real-world objects. The CARLA simulator enables a user to control a vehicle in an open-world simulator and monitor the output of different sensors. The sensors generate 3D bounding boxes for objects on the road, which includes other vehicles, pedestrians, road signs, etc. Similar technology is used by vehicle manufacturers (e.g., Audi [16], Honda [21]).

(3) **3D oriented bounding boxes from ScanNet:** We used a dataset of 3D oriented bounding boxes of objects in the ScanNet dataset[8]. ScanNet contains aerial images of objects in domestic settings. The dataset chosen has been used for validation of a model identifying 3D oriented bounding boxes [45].

Platform: All experiments (unless otherwise stated) were run on two t2.xlarge Amazon EC2 instances with 4 vCPUs and 16GB of RAM, placed in different zones (US East and West). The network bandwidth between them was measured to be around 40–60MB per second using *iperf*[9].

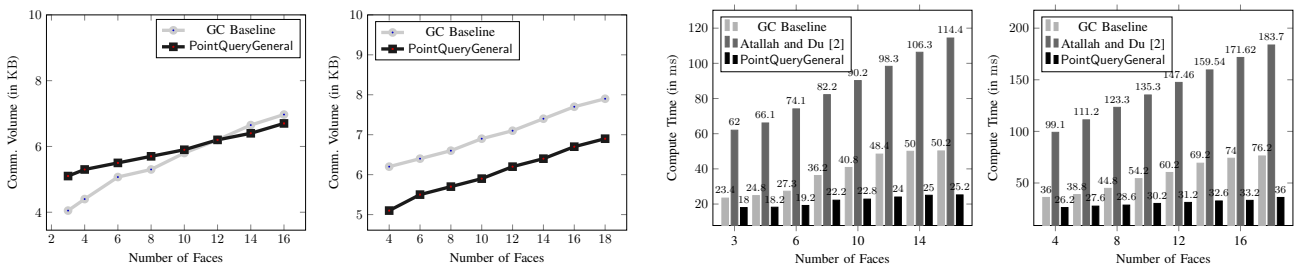
9.1. Microbenchmarks

Private Point Queries: We compared our private point query protocol PointQueryGeneral with a garbled-circuit implementation and the point inclusion protocol of Atallah and Du [2]. The benchmark is over random polytopes with different numbers of vertices/faces. The objective is to demonstrate how our protocol scales with the complexity of the polytopes input to the protocol as compared to the garbled-circuit baseline. Fig. 12 shows the results:

(1) **Communication Volume:** Fig. 12a and Fig. 12b show how communication volume scales with the number of faces of the input polytope in $\mathbb{R}^2$ and $\mathbb{R}^3$, respectively. The communication cost of PointQueryGeneral scales better than the GC baseline and for polygons with more than 10 edges, PointQueryGeneral has a lower communication volume compared to the baseline. In $\mathbb{R}^3$, PointQueryGeneral has lower communication volumes for all polytopes with 4–18 faces.
The GC baseline and PointQueryGeneral both have lower communication requirements than the protocol of Atallah and Du [2] by roughly two orders of magnitude. This is expected as the cost of their protocol depends on λ and κ (see Table 1). For 2048-bit keys and computational security parameter $\lambda = 80$, the communication cost of the point inclusion protocol of Atallah and Du [2] for a 4 sided polygon is 640 KB in comparison to 4.4 KB for PointQueryGeneral. PointQueryGeneral communication cost also scales better han the protocol of Atallah and Du [2]; a full comparison can be found in the Appendix.

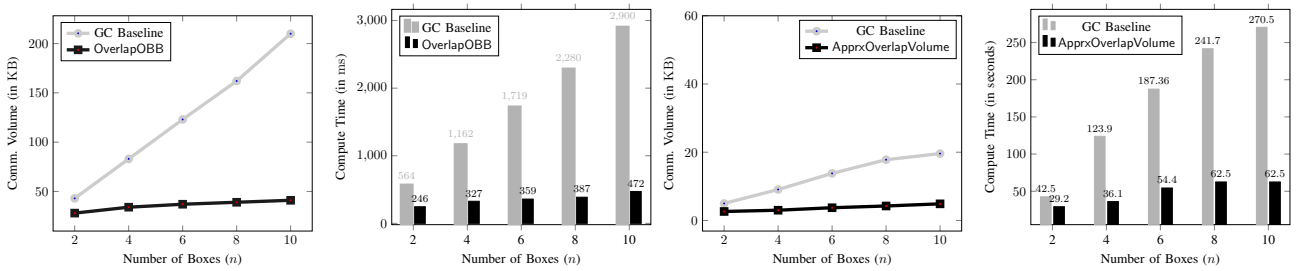
(2) **Compute time:** Fig. 12c and Fig. 12b show how compute time scales with the number of faces of the input polytope in $\mathbb{R}^2$ and $\mathbb{R}^3$, respectively. For polygons in $\mathbb{R}^2$, PointQueryGeneral is 1.3–2$\times$ faster than the GC baseline, and in $\mathbb{R}^3$, PointQueryGeneral is 1.4–2.1$\times$ faster. Both PointQueryGeneral and the GC baseline outperform the construction of Atallah and Du [2] where signficant time is spend in encryption and decryption; PointQueryGeneral is $2.7-5.1\times$ faster than the construction of Atallah and Du [2].

6. https://github.com/emp-toolkit/emp-zk
7. https://github.com/emp-toolkit/emp-sh2pc
8. https://github.com/qq456cvb/CanonicalVoting
9. https://iperf.fr/



(a) 2D: Comm. vs. Num of Faces (b) 3D: Comm. vs. Num of Faces (c) 2D: Time vs. Num of Faces (d) 3D: Time vs. Num of Faces

Figure 12: Comparison of PointQueryGeneral with a garbled circuit baseline. The dataset includes randomly-generated polytopes with different number of faces. For polygons (in 2D) with more than 10 faces, PointQueryGeneral has lower communication volume. In 3D, PointQueryGeneral has lower communication for polytopes with up to 18 faces. PointQueryGeneral outperforms the baseline in terms of compute time. In 2D, PointQueryGeneral is $\mathbf{1.3-2}\times$ faster and in 3D PointQueryGeneral is $\mathbf{1.4-2.1}\times$ faster than the baseline. Both PointQueryGeneral and the GC baseline are faster than the construction of Atallah and Du [2] and the communication costs of both protocols are lower by roughly two orders of magnitude (see Fig. 17 in the Appendix for more details).



(a) Comm. vs. Num of Boxes (b) Comp. vs. Num of Boxes (c) Comm. vs. Num of Boxes (d) Comp. vs. Num of Boxes

Figure 13: Communication volume and compute time of OverlapOBB and ApprxOverlapVolume when detecting overlaps and approximating volume of overlaps respectively for a set of oriented boxes. With set size $n = 10$, OverlapOBB is $\mathbf{6.1}\times$ faster and has $\mathbf{5.1}\times$ lower communication volume. With $n = 10$, ApprxOverlapVolume is $\mathbf{4.17}\times$ faster and requires $\mathbf{4}\times$ lower communication.

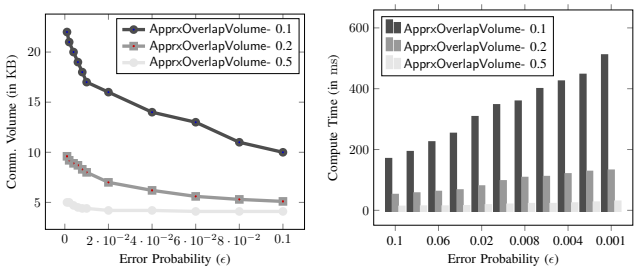
9.2. CARLA Axis-Aligned Boxes

We used the CARLA simulator to generate views of the same road intersection from different vehicles (at that intersection) and compared the objects in their views. The simulator generates 3D axis-aligned bounding boxes for other vehicles, road signs, pedestrians, and traffic signs.

Measure	Comm	Comp
OverlapAABB	42 KB	2.80 ms
GC baseline	51.0 KB	11.60 ms
Atallah and Du [2]	11.5 MB	496 ms

TABLE 3: Communication and computation costs of OverlapAABB, a garbled-circuit baseline and the protocol of Atallah and Du [2] when detecting if two axis-aligned boxes overlap. OverlapAABB is $\mathbf{4.2}\times$ faster than the GC baseline and $\mathbf{165}\times$ faster than the construction of Atallah and Du [2].

Table 3 shows how OverlapAABB compares with a garbled-circuit baseline and the construction of Atallah and Du [2] when detecting if two axis-aligned boxes overlap. We note that Atallah and Du [2] only provide a protocol for detecting overlaps of convex polygons which we extend to polytopes and use here in the benchmarking. The GC baseline implements a simple process where each vertex of a party's bounding box is checked to determine whether it lies inside the other party's bounding box. If any of the vertices lie inside, the bounding boxes overlap. OverlapAABB is roughly $4.2\times$ faster than the baseline and has 20% lower communication costs. Compared to the construction of Atallah and Du [2], OverlapAABB is $165\times$ faster; this is expected since the cost of OverlapAABB scales linearly in the number of dimensions in contrast to exponentially-scaling costs for their protocol (see Table 1)



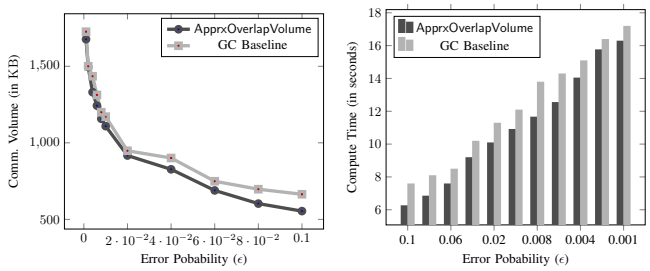
(a) Comm. vs. Error Prob. (b) Time vs. Error Prob.

Figure 14: Communication volume and compute time of ApprxOverlapVolume when approximating the volume of overlap of two axis aligned bounding boxes for error probability ϵ. The plots correspond to the slack parameter, $\delta = 0.1$, $\delta = 0.2$ and $\delta = 0.5$.

Measuring Volume of Overlap: We ran experiments to estimate the communication and compute costs of approximating volume of overlap between two axis-aligned bounding boxes using ApprxOverlapVolume. Fig. 14a shows how communication volume scales with the error probability ϵ for different values of the slack parameter δ.

As expected, the communication volume scales inversely with the error probability. For $\epsilon = 0.001$, and $\delta = 0.1$, the protocol requires 22 KB of communication.

Fig. 14b shows how compute time scales with the error probability. Similar to the communication volume, the compute time scales inversely with ϵ. For $\epsilon = 0.001$ and $\delta = 0.1$, the protocol requires 508 ms to approximately compute the volume of the overlapping region.



(a) Comm. vs. Error Prob. (b) Time vs. Error Prob.

Figure 15: Communication volume and compute time when applying ApprxOverlapVolume to approximate the volume of overlap of two oriented bounding boxes compared to a garbled circuit baseline for error probability ϵ. $\delta = 0.1$. ApprxOverlapVolume is $\mathbf{1.2}\times$ faster than the baseline and requires 20% less communication.

9.3. ScanNet Bounding Boxes

ScanNet is an annotated dataset of 3D objects in indoor domestic scenes. Recent work [45] on generating bounding boxes for 3D objects has been evaluated on the dataset. We have used the ground truth data used for validation in [45] as inputs to OverlapOBB and ApprxOverlapVolume. The baseline is a garbled circuit implementation of the separating axis theorem (see Sec. 3), and the protocol by Atallah and Du [2].

Measure	Comm	Comp
OverlapOBB	26.0 KB	211 ms
GC baseline	30.0 KB	280 ms
Atallah and Du [2]	11.5 MB	496 ms

TABLE 4: Comparison of communication and computation costs of OverlapOBB with a garbled circuit baseline when detecting if two oriented bounding boxes overlap. OverlapOBB is $\mathbf{1.4}\times$ faster than the GC baseline and $\mathbf{2.3}\times$ faster than the construction of Atallah and Du [2] and requires $\mathbf{443}\times$ lower communication volume.

Detecting Overlaps: Table 4 compares OverlapOBB with the baseline in terms of communication volume and compute times. OverlapOBB is roughly 40% faster than the baseline and requires 15% lower communication volume. Compared to Atallah and Du [2], OverlapOBB is $2.3\times$ faster due to the use of cheaper primitives and requires roughly $443\times$ lower communication volume.

OverlapAABB **vs.** OverlapOBB: OverlapOBB can also be directly applied to axis-aligned bounding boxes. While doing so, the communication volume and compute times are going to be the same as those reported in Table 4. This is because OverlapOBB is agnostic to the orientation of the box. Interestingly, OverlapAABB and OverlapOBB present a trade-off in communication volumes and compute time. When using OverlapAABB for axis-aligned boxes, the compute time is optimized since we do not need to compute scalar products to check if vertices of one box lies within the other (see Sec. 7.1 for more details). On the other hand, OverlapOBB needs to compute the scalar product to check for vertex inclusion, but due to the use of a communication-efficient VOLE protocol to implement this, the communication volume of OverlapOBB is lower.

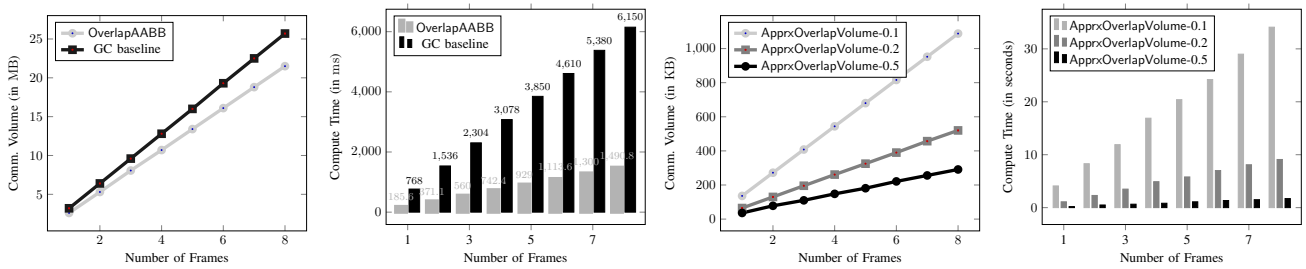
Measuring Volume of Overlap: Fig. 15 compares ApprxOverlapVolume when approximating the volume of overlap of two oriented boxes with a garbled circuit baseline. The slack parameter $\delta = 0.1$; similar trends are observed for other values of δ and the results are omitted. OverlapOBB requires roughly 20% lower communication volume and is $1.2\times$ faster than the baseline.

Scaling to Multiple Boxes: In realistic settings, Alice and Bob will observe multiple objects (boxes) in their immediate surrounding. For instance, vehicles at an intersection will observe multiple objects which includes other vehicles, pedestrians, road signs etc. Thus, the box corresponding to each such object observed by Alice must be compared with all the boxes of the objects observed by Bob. In our experiment, this corresponds to Alice providing as input a bounding box, while Bob provides a set of n boxes. Of course, the trivial solution is running each protocol n times independently, but as described in Sec. 6 and Sec. 7, our protocols can be optimized to batch the n instances together for better communication and compute costs. This is applicable to OverlapOBB and ApprxOverlapVolume for oriented boxes.

Fig. 13a and Fig. 13b show how communication volume and compute time scale with n, the number of boxes input by Bob in OverlapOBB. With 10 boxes to compare with Alice's input box, the baseline requires up to $5.1\times$ more communication volume compared to OverlapOBB. The improvement comes from the fact that the most expensive part of the protocol in OverlapOBB i.e., computing vector dot products is implemented using the communication efficient VOLE protocol. The optimization batching dot-product computations using VOLE across all boxes input by Bob yields not only the improvements over the baseline, but also shows at least a $4\times$ improvement in communication costs over a naive repetition of OverlapOBB for each of the input boxes.

Fig. 13b shows how compute time scales with the number of boxes compared. OverlapOBB is up to $6.1\times$ faster than the baseline as the VOLE protocol used relies only on cheap symmetric key primitives. On the other hand, in the case of the baseline implementing the separating axis theorem, the garbled circuit implements several expensive steps which includes computing vector cross products to generate candidate axes and computing projections along these axes.

Similar improvements are also observed for ApprxOverlapVolume. Fig. 13c shows that the communication cost of ApprxOverlapVolume scales more gracefully than the baseline with the number of boxes compared. With 10 boxes to compare, the baseline incurs $4\times$ higher communication costs. Fig. 13d shows that compute time of ApprxOverlapVolume scales better with the number of boxes than the baseline. As



(a) Comm. vs. Num of Frames (b) Comp. vs. Num of Frames (c) Comm. vs. Num of Frames (d) Comp. vs. Num of Frames

Figure 16: Communication volume and compute time of OverlapAABB and ApprxOverlapVolume when detecting overlaps and approximating volume of overlaps, respectively, for objects in frames derived from the A2D2 dataset. When comparing objects in a frame observed by a vehicles with objects in frames observed by 8 other vehicles, OverlapAABB is $4.1\times$ faster and requires 20% less communication. Approximating the volume of overlap between objects in the frames requires up to 1 MB of communication with error probability $\epsilon = 0.01$ and slack parameter $\delta = 0.1$, and takes around 34 seconds of compute time.

expected, using VOLE to perform vector dot products is significantly more efficient than performing these operations inside the garbled circuit. With 10 boxes to compare, ApprxOverlapVolume is $4.17\times$ faster than the baseline. Additionally, compared to naively repeating ApprxOverlapVolume for each of the n boxes input by Bob, batching the dot-product computations across all n instances using VOLE results in $3.4\times$ lower communication cost and $3.2\times$ lower compute time.

9.4. AUDI A2D2 Dataset

A2D2 [16] is an autonomous driving dataset released by AUDI. Among other things, the dataset provides 3D bounding boxes for objects observed on the road. These objects include other vehicles, traffic signals, pedestrians, buildings, etc. The dataset includes more than $12{,}000$ *frames*, where each frame is a camera snapshot of the road observed from a specific vehicle at a given point in time, and includes annotated 3D axis-aligned bounding boxes for objects in the frame. For benchmarking our protocols, we have randomly sampled 100 frames out of the available $12{,}000$, and we run pairwise comparisons between objects in these frames. This represents two vehicles observing frames of the road using their cameras and LIDAR sensors, and comparing and validating outputs. We have estimated compute times and communication volume for detecting overlaps and approximating the overlap volume.

Platform: To simulate realistic settings, we ran our experiments on low-resource setups available on AWS. Specifically, we used t2.small EC2 instances available on AWS with 1 available vCPU and 2 GiB of RAM, running Linux. In contrast, there are proposals suggesting that future smart vehicles will be equipped with powerful processors and advanced graphics capabilities[10]. Thus, our experiments may in fact underestimate performance.

Detecting Overlaps Between Objects in Frames: Fig. 16a and Fig. 16b show how communication and compute time scales with OverlapAABB when a frame observed by one particular vehicle is compared with different number of frames, each observed by a different vehicle. The baseline is a garbled-circuit implementation detecting if two axis-aligned bounding boxes overlap. The frames in our dataset typically contain 8 detected road objects; i.e., each frame contains 8 axis-aligned bounding boxes. As expected, OverlapAABB is $4.1\times$ faster than the baseline and requires 20% lower communication costs.

Approximating Volume of Overlap: Fig. 16c and Fig. 16d show how communication volume and compute time scales when approximating the volume of object overlap in frames captured by different vehicles. For the experiments, the error probability $\epsilon = 0.01$. When comparing 8 frames (captured by 8 other vehicles) with a frame captured by a particular vehicle, the compute time required for approximating volume of overlap between objects is roughly 34 seconds with the slack parameter $\delta = 0.1$. The total communication volume is around 1MB.

10. Conclusion

This paper presents cryptographic protocols for privately computing if two regions in Euclidean space overlap and approximating the volume of the overlapping region. These protocols can aid validation of sensor outputs (as part of computer vision systems) of mutually untrusting parties without revealing sensitive information, and can be used to detect faults, help in better decision-making, etc. From an intellectual perspective, this paper takes a step forward in building efficient protocols for secure computational geometry with real-world applications to, e.g., connected autonomous vehicles (CAVs). The protocols rely on cheap cryptographic primitives and feature reasonable communication requirements and compute times. The protocols have been benchmarked on real-world datasets and outperform garbled-circuit-based 2PC baselines and tailormade constructions for 2D.

11. Acknowledgments

This research was supported in part by NSF grant 2113345, NIFA grant 2021-67021-34252, and ARO grant W911NF-17-1-0370. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, NIFA, ARO, or the U.S. Government.

10. https://developer.nvidia.com/blog/now-available-drive-agx-orin-with-drive-os-6/

References

[1] National Highway Traffic Safety Administration et al. Traffic safety facts: 2007 data: Pedestrians. *Annals of Emergency Medicine*, 53(6):824, 2009.

[2] Mikhail J. Atallah and Wenliang Du. Secure multi-party computational geometry. In *Workshop on Algorithms and Data Structures*. Springer, 2001.

[3] Karl Bringmann and Tobias Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. *Computational Geometry*, 43(6-7):601–610, 2010.

[4] Anrin Chakraborti, Giulia Fanti, and Michael K. Reiter. Distance-aware private set intersection. In *32nd USENIX Security Symposium (USENIX Security 23)*, 2023. URL https://www.usenix.org/system/files/sec23summer_333-chakraborti-prepub.pdf.

[5] Hao Chen, Kim Laine, Rachel Player, and Yuhou Xia. High-precision arithmetic in homomorphic encryption. In *Cryptographers' Track at the RSA Conference*, pages 116–136. Springer, 2018.

[6] Christopher Clapham and James Nicholson. The concise oxford dictionary of mathematics (4 ed.). 2009.

[7] H.S.M Coxeter. Regular polytopes 3 ed. 1973.

[8] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[9] David Elliott, Walter Keen, and Lei Miao. Recent advances in connected and automated vehicles. *journal of traffic and transportation engineering (English edition)*, 6(2):109–131, 2019.

[10] Wilfried Elmenreich. An introduction to sensor fusion. *Vienna University of Technology, Austria*, 502:1–28, 2002.

[11] Daniel J Fagnant and Kara Kockelman. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181, 2015.

[12] Pierre-Alain Fouque, Jacques Stern, and Geert-Jan Wackers. Cryptocomputing with rationals. In *International Conference on Financial Cryptography*, pages 136–146. Springer, 2002.

[13] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19, 2004.

[14] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. Structure-aware private set intersection, with applications to fuzzy matching. In *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part I*, pages 323–352. Springer, 2022.

[15] Ewgenij Gawrilow and Michael Joswig. `polymake`: a framework for analyzing convex polytopes. In *Polytopes—combinatorics and computation (Oberwolfach, 1997)*, volume 29 of *DMV Sem.*, pages 43–73. Birkhäuser, Basel, 2000.

[16] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, Tiffany Fernandez, Martin Jänicke, Sudesh Mirashi, Chiragkumar Savani, Martin Sturm, Oleksandr Vorobiov, Martin Oelker, Sebastian Garreis, and Peter Schuberth. A2D2: Audi Autonomous Driving Dataset. 2020. URL https://www.a2d2.audi.

[17] Stefan Aric Gottschalk. *Collision queries using oriented bounding boxes*. The University of North Carolina at Chapel Hill, 2000.

[18] Adam Groce, Peter Rindal, and Mike Rosulek. Cheaper private set intersection via differentially private leakage. *Proceedings on Privacy Enhancing Technologies*, 2019:25–6, 2019.

[19] Jianhua He, Zuoyin Tang, Xiaoming Fu, Supeng Leng, Fan Wu, Kaisheng Huang, Jianye Huang, Jie Zhang, Yan Zhang, Andrew Radford, et al. Cooperative connected autonomous vehicles (cav): research, applications and challenges. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–6. IEEE, 2019.

[20] Anthony J Healey and Douglas P Horner. Collaborative vehicles in future naval missions, obstacle detection and avoidance. *In Proceedings of the IFAC Conference on Modelling and Control of Marine Craft*, 2006.

[21] Honda. Hdd hri driving dataset, 2022. URL https://usa.honda-ri.com/hdd.

[22] Samuel Hornus. A review of polyhedral intersection detection and new techniques. Research Report RR-8730, INRIA, 2015. URL https://hal.inria.fr/hal-01157239v1/document. hal-01157239v1.

[23] Bailey Kacsmar, Basit Khurram, Nils Lukas, Alexander Norton, Masoumeh Shafieinejad, Zhiwei Shang, Yaser Baseri, Maryam Sepehri, Simon Oya, and Florian Kerschbaum. Differentially private two-party set operations. In *2020 IEEE European Symposium on Security and Privacy*, pages 390–404, September 2020. doi: 10.1109/EuroSP48549.2020.00032.

[24] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, page 241–257, 2005. doi: 10.1007/11535218_15. URL https://doi.org/10.1007/11535218_15.

[25] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer, 2008.

[26] Pantelis Kopelias, Elissavet Demiridi, Konstantinos Vogiatzis, Alexandros Skabardonis, and Vassiliki Zafiropoulou. Connected & autonomous vehicles–environmental impacts–a review. *Science of the total environment*, 712:135237, 2020.

[27] Changliu Liu, Chung-Wei Lin, Shinichi Shiraishi, and Masayoshi Tomizuka. Distributed conflict resolution for connected autonomous vehicles. *IEEE Transactions on Intelligent Vehicles*, 3(1):18–29, 2017.

[28] Silvio Micali, Oded Goldreich, and Avi Wigderson.

How to play any mental game. In *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*, pages 218–229. ACM, 1987.

[29] Bin Mu and Spiridon Bakiras. Private proximity detection for convex polygons. 2013. URL https://doi.org/10.1145/2486084.2486087.

[30] Bin Mu and Spiridon Bakiras. Private proximity detection for convex polygons. *Tsinghua Science and Technology*, 21(3):270–280, 2016.

[31] University of California San Diego. Collaborative intelligence in smart and connected vehicles, 2023. URL http://cwc.ucsd.edu/research/collaborative-intelligence-smart-and-connected-vehicles.

[32] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 797–812, San Diego, CA, August 2014. USENIX Association. ISBN 978-1-931971-15-7. URL https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/pinkas.

[33] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium*, pages 515–530, August 2015.

[34] Benny Pinkas, Mike Rosulek, Ni Trieu, and A. Yanai. Psi from paxos: Fast, malicious private set intersection. In *Advances in Cryptology –- EUROCRYPT 2020*, volume 12106 of *Lecture Notes in Computer Science*, pages 739–767, 2020.

[35] Jurek Z Sasiadek. Sensor fusion. *Annual Reviews in Control*, 26(2):203–228, 2002.

[36] Steven E Shladover, Dongyan Su, and Xiao-Yun Lu. Impacts of cooperative adaptive cruise control on freeway traffic flow. *Transportation Research Record*, 2324(1):63–70, 2012.

[37] Victor Shoup et al. Ntl: A library for doing number theory, 2001.

[38] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *2015 IEEE Symposium on Security and Privacy*, pages 411–428. IEEE, 2015.

[39] Alireza Talebpour and Hani S Mahmassani. Influence of connected and autonomous vehicles on traffic flow stability and throughput. *Transportation Research Part C: Emerging Technologies*, 71:143–163, 2016.

[40] Elisabeth Uhlemann. Time for autonomous vehicles to connect [connected vehicles]. *IEEE vehicular technology magazine*, 13(3):10–13, 2018.

[41] Erkam Uzun, Simon P Chung, Vladimir Kolesnikov, Alexandra Boldyreva, and Wenke Lee. Fuzzy labeled private set intersection with applications to private real-time biometric search. In *USENIX Security Symposium*, pages 911–928, 2021.

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

[43] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.

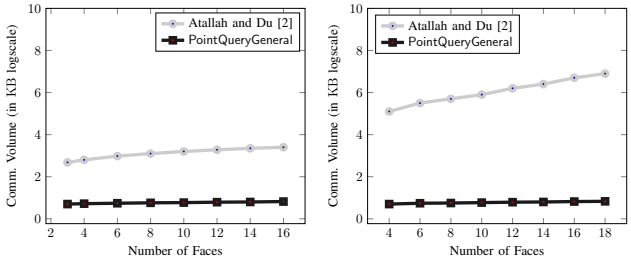
[44] Lanhang Ye and Toshiyuki Yamamoto. Evaluating the impact of connected and autonomous vehicles on traffic safety. *Physica A: Statistical Mechanics and its Applications*, 526:121009, 2019.

[45] Yang You, Zelin Ye, Yujing Lou, Chengkun Li, Yong-Lu Li, Lizhuang Ma, Weiming Wang, and Cewu Lu. Canonical voting: Towards robust oriented bounding box detection in 3d scenes. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[46] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 220–250. Springer, 2015.

[47] Hui Zhu, Fengwei Wang, Rongxing Lu, Fen Liu, Gang Fu, and Hui Li. Efficient and privacy-preserving proximity detection schemes for social applications. *IEEE Internet of Things Journal*, 5(4): 2947–2957, 2018. doi: 10.1109/JIOT.2017.2766701.

Appendix



(a) 2D: Comm. vs. Num of Faces (b) 3D: Comm. vs. Num of Faces

Figure 17: Communication cost (in logscale) of the protocol of Atallah and Du [2] when processing polytopes of different complexities. PointQueryGeneral has more than two orders of magnitude lower communication requirements, and scales more gracefully than the protocol of Atallah and Du [2]

1. Proofs of Theorems

Theorem. *Assuming that there is a protocol that realizes* $\mathcal{F}_{ole}$ *with* $\mathrm{O}(\lambda)$ *bits of communication,* DotProdCheck *realizes* $\mathcal{F}_{dot}^{d}$ *with* $\mathrm{O}(d\lambda)$ *bits communication.*

Proof. The communication cost of DotProdCheck is straightforward: Alice and Bob call $\mathcal{F}_{ole}$ d times, and overall this step costs $\mathrm{O}(d\lambda)$ bits of communications. In addition, Alice and Bob compute a garbled circuit with d inputs, which also costs $\mathrm{O}(d\lambda)$ bits of communication. The total communication cost of the protocol is $\mathrm{O}(d\lambda)$.

Security: The security of the scheme relies on the argument that if there is a protocol, π realizing $\mathcal{F}_{ole}$ and there is a PPT simulator, Sim_{π} that can indistinguishably simulate Alice's and Bob's views in the ideal world simulation of the real world execution of π, then there is a PPT simulator Sim that can indistinguishably simulate Alice's and Bob's views in the ideal world simulation of the real world execution of DotProdCheck.

Simulating Bob*'s view:* In the real world execution of DotProdCheck, Bob does not receive any output, and only receives the intermediate results from π when computing s. In the ideal world, for $i \in [1, d]$, Sim sends Bob's inputs b_i and r_i to $\mathcal{F}_{ole}$ along with randomly sampled elements $r'_i \stackrel{\$}{:=} \mathbb{F}$. Assuming that there is a protocol realizing $\mathcal{F}_{ole}$ that can be indistinguishably simulated, Bob's view in the ideal world simulation of DotProdCheck can be indistinguishably simulated by Sim.

Simulating Alice*'s view:* In the real world execution of DotProdCheck, Alice receives s. In the ideal world execution, for each $i \in [1, d]$, Sim sends Alice's input a_i to $\mathcal{F}_{ole}$. Additionally, Sim samples random elements $b'_i \stackrel{\$}{:=} \mathbb{F}$ and r'_i and sends as inputs to $\mathcal{F}_{ole}$. Sim provides output of $\mathcal{F}_{ole}$, $a_i b'_i + r'_i$, to Alice. Note that $a_i b'_i + r'_i \stackrel{\$}{:=} \mathbb{F}$ is indistinguishable from the output in the real world execution of DotProdCheck, $a_i b_i + r_i \stackrel{\$}{:=} \mathbb{F}$ since $r_i \stackrel{\$}{:=} \mathbb{F}$.

Subsequently, Sim generates a garbled circuit, which takes as input s from Alice. Additionally, if Alice's output from DotProdCheck is TRUE, Sim's input into the garbled circuit is some $r' \in [s - \mathsf{upper}, s - \mathsf{lower}] \in \mathbb{F}$ and if Alice's output from DotProdCheck is FALSE, Sim's input into the garbled circuit is some $r' \notin [s - \mathsf{upper}, s - \mathsf{lower}] \in \mathbb{F}$. Since a garbled circuit can realize any function securely, any secure implementation of a garbled circuit ensures security of this step and makes the real world execution and ideal world simulation indistinguishable. □

Theorem. *Assuming that there is a protocol realizing $\mathcal{F}_{vole}$ for ϕ-length vectors with $\mathrm{O}(\phi\lambda)$ bits of communication,* BlindedMatrixMultiply *realizes $\mathcal{F}^d_{bmul}$ with $\mathrm{O}(\phi d\lambda)$ bits of communication.*

Proof. The communication cost of BlindedMatrixMultiply is straightforward: Alice and Bob call $\mathcal{F}_{vole}$ d times with vectors of length ϕ. The total communication cost of this step is $\mathrm{O}(\phi d\lambda)$ bits of communication. The rest of the steps are non-interactive and do not require any communication.

Security: The security of the scheme relies on the argument that if there is a protocol, π that securely realizes $\mathcal{F}_{vole}$ and there is a PPT simulator, Sim_π that can indistinguishably simulate Alice's and Bob's views in the ideal world simulation of the real world execution of π, then there is a PPT simulator Sim that can indistinguishably simulate Alice's and Bob's views in the ideal world simulation of the real world execution of BlindedMatrixMultiply.

Simulating Bob*'s view:* In the real world execution, Bob does not receive any output from BlindedMatrixMultiply and only receives the intermediate results when engaging in d calls to a protocol π securely realizing $\mathcal{F}_{vole}$. In the ideal world, for each $j \in [1, d]$, Sim's sends Bob's inputs $F[* : j]$ and $R[* : j]$ along with a random element $a'_j \stackrel{\$}{:=} \mathbb{F}$ to $\mathcal{F}_{vole}$. Assuming that there is a protocol realizing $\mathcal{F}_{vole}$ that can be indistinguishably simulated, Bob's view in the ideal world simulation of BlindedMatrixMultiply can be indistinguishably simulated by Sim.

Simulating Alice*'s view:* In the real world execution of BlindedMatrixMultiply, for each $j \in [1, d]$, Alice's output is a vector $[a_j c_{1,j} + r_{1,j}, \ldots, a_j c_{\phi,j} + r_{\phi,j}]$. In the ideal world simulation, for each $j \in [1, d]$, Sim calls $\mathcal{F}_{vole}$ with inputs: i) a_j, and ii) the vectors $[c'_{1,j}, \ldots, c_{\phi,j}] \stackrel{\$}{:=} \mathbb{F}^\phi$ and $[r'_{1,j}, \ldots, r'_{\phi,j}] \stackrel{\$}{:=} \mathbb{F}^\phi$. Sim provides the output of $\mathcal{F}_{vole}$, $[a_j c'_{1,j} + r'_{1,j}, \ldots, a_j c'_{\phi,j} + r'_{\phi,j}]$ to Alice. Since, $[a_j c'_{1,j} + r'_{1,j}, \ldots, a_j c'_{\phi,j} + r'_{\phi,j}] \stackrel{\$}{:=} \mathbb{F}^\phi$, it is indistinguishable from the real world output $[a_j c_{1,j} + r_{1,j}, \ldots, a_j c_{\phi,j} + r_{\phi,j}] \stackrel{\$}{:=} \mathbb{F}^\phi$ as $r_{1,j}, \ldots, r_{\phi,j} \stackrel{\$}{:=} \mathbb{F}$. Thus, Alice's view can be indistinguishably simulated in the ideal world execution by Sim. □

2. Comparison with Atallah and Du [2]

We have run experiments to evaluate how the communication costs of PointQueryGeneral compares to the protocol of Atallah and Du [2] for polygons/polytopes of different complexities (i.e., number of edges/faces). The polygons and polytopes are sampled from the dataset of randomly-generated polytopes described in Sec. 9. Fig. 17 describes the result. The communication cost of PointQueryGeneral is almost two orders of magnitude lower than the protocol of Atallah and Du [2]. This is expected since the communication cost of PointQueryGeneral is asymptotically lower by a factor of the key size of the additively homomorphic encryption scheme used in their scheme, κ (see Table 1 for reference).