Privacy-preserving Density-based Clustering[*]

Beyza Bozdemir
EURECOM
Sophia Antipolis, France
beyza.bozdemir@eurecom.fr

Sébastien Canard
Applied Crypto Group, Orange Labs
Caen, France
sebastien.canard@orange.com

Orhan Ermis
EURECOM
Sophia Antipolis, France
orhan.ermis@eurecom.fr

Helen Möllering[†]
Technical University of Darmstadt
Darmstadt, Germany
moellering@encrypto.cs.tu-darmstadt.de

Melek Önen
EURECOM
Sophia Antipolis, France
melek.onen@eurecom.fr

Thomas Schneider
Technical University of Darmstadt
Darmstadt, Germany
schneider@encrypto.cs.tu-darmstadt.de

ABSTRACT

Clustering is an unsupervised machine learning technique that outputs clusters containing similar data items. In this work, we investigate privacy-preserving density-based clustering which is, for example, used in financial analytics and medical diagnosis. When (multiple) data owners collaborate or outsource the computation, privacy concerns arise. To address this problem, we design, implement, and evaluate the first practical and fully private density-based clustering scheme based on secure two-party computation. Our protocol privately executes the DBSCAN algorithm without disclosing any information (including the number and size of clusters). It can be used for private clustering between two parties as well as for private outsourcing of an arbitrary number of data owners to two non-colluding servers. Our implementation of the DBSCAN algorithm privately clusters data sets with 400 elements in 7 minutes on commodity hardware. Thereby, it flexibly determines the number of required clusters and is insensitive to outliers, while being only factor 19x slower than today's fastest private K-means protocol (Mohassel et al., PETS'20) which can only be used for specific data sets. We then show how to transfer our newly designed protocol to related clustering algorithms by introducing a private approximation of the TRACLUS algorithm for trajectory clustering which has interesting real-world applications like financial time series forecasts and the investigation of the spread of a disease like COVID-19.

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**; • **Computing methodologies** → *Cluster analysis.*

KEYWORDS

Private Machine Learning, Clustering, Secure Computation

1 INTRODUCTION

The availability of vast amounts of data and cloud computing power nowadays has lead to hype around machine learning (ML). Supervised ML techniques like neural networks use labeled data records (i.e., known input-output pairs) to train a model later utilized, e.g., for the classification of new records. In contrast, unsupervised ML techniques have no "training" phase and aim at detecting unknown patterns and structures in the unlabeled input data. Clustering is a widespread unsupervised ML technique that partitions data into groups of elements with similar properties. It has many privacy-critical applications where sensitive business or personal data must be protected spanning from financial analytics over market research to medical diagnoses [1, 26, 56].

For such analyses, data from several sources is often needed, for example, from competing (investment) banks or insurance companies to detect suspicious behavior [63] or from several hospitals to get a diverse data set that is not biased due to diverse backgrounds and demographics. Generally, clustering more data from several sources commonly enhances the quality of analyses. Moreover, it can be attractive to outsource computation and data due to high requirements for storage and costly computation. However, in both collaborative analyses and when outsourcing computation, the sensitive input data requires protection against untrusted servers and other data owners. Secure computation techniques can protect against these parties to preserve data privacy.

Several optimized private clustering protocols using secure computation have already been proposed for the well-known K-means algorithm [49, 71]. However, K-means is relatively simple and can only cluster specific data. It only detects convexly shaped clusters such that it is only suitable for specific data collections. Additionally, the number of clusters K needs to be determined in advance which requires domain knowledge. Access to just a subset of the input data makes it difficult to determine K. Also, K-means does not include the notion of noise and its result is highly sensitive to outliers as it has to assign every input to a cluster. Hence, even if a record does not fit into any cluster, it will be assigned to the least distant one and heavily affect this cluster's centroid (i.e., the mean of all assigned elements).

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a more flexible clustering algorithm introduced by Ester et al. in [20] to address the weaknesses of previously known clustering algorithms like K-means. DBSCAN is able to detect arbitrarily shaped clusters. Additionally, the number of clusters is flexibly determined such that it fits to the data. The algorithm is insensitive to outliers and specifically marks them as noise. Fig. 1 demonstrates the advantages of DBSCAN over K-means on four different data sets.

[*]Please cite the version of this paper published at 16th ACM ASIA Conference on Computer and Communications Security (ACM ASIACCS 2021) [13].
[†]Contact Author.

TRACLUS [45], an extension of DBSCAN, is specifically designed for clustering trajectories (i.e., sequences of multidimensional points). Trajectories intuitively do not form convex clusters. Furthermore, as trajectories often have different lengths, K-means cannot straightforwardly cluster them, because it requires the same fixed number of parameters for all inputs.

Private trajectory clustering can be a valuable mean for financial time series forecasts [29, 54] which are used by investors for decision making and rely on the input of sensitive business data. Additionally, it can be used for the privacy-preserving analysis of location data collected by telecommunication providers to optimize the services of the travel industry by determining typical traveling routes. Another interesting application is the analysis of the movement of (infected) people to control the spread of COVID-19 while maintaining privacy in accordance to regulations like the GDPR. In this context, it enables to detect people who used similar routes while the service provider cannot learn individual movements. Moreover, policy makers could use privacy-preserving trajectory clustering to privately determine if people comply to social distancing in the current pandemic before deciding about further regulations.

Need for Full Privacy-preservation. As we will discuss in §2, many previous "private" clustering protocols leak information beyond the output. We now demonstrate why leaking such intermediate information can cause a severe privacy breach using an example from [47].

Liu et al. [47] demonstrate how leaking which elements are neighbors, as done in [43], can be used to approximate data records of other parties. We depict the attack in Fig. 2. Let us consider two data owners, Alice and Bob, who hold a horizontally partitioned data set, i.e., different data records that must not be leaked to each other. Bob holds three data records B_1, B_2, and B_3 that are too far from each other (i.e., their mutual distances are greater than ϵ, cf. §3.1) to create a cluster themselves. However, when Alice holds a data record A that lies exactly in the intersection of the neighborhoods of B_1, B_2, and B_3 indicated by the circles with radius ϵ, Bob learns that all three of his data records are neighbors of the same element of Alice. From this information and if this intersection is small, Bob can accurately approximate A.

To summarize, such information leakage beyond the clustering output can cause serious privacy infringements, and hence should be avoided. In this work, we provide the first fully privacy-preserving DBSCAN algorithm that does not leak any information beyond the output of the clustering.

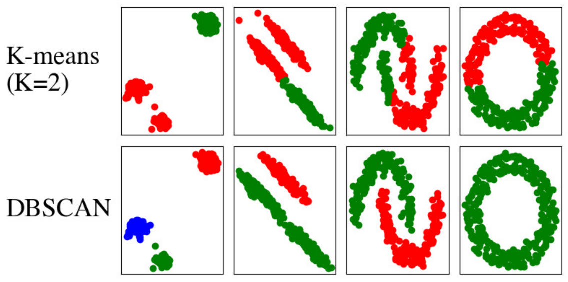


Figure 1: Comparison of clustering results with K-means and DBSCAN.

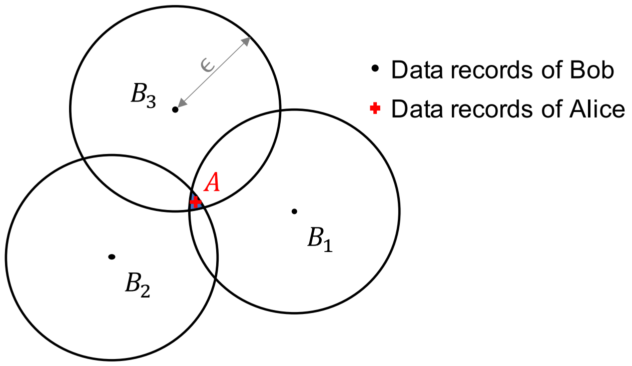


Figure 2: Approximating data records of other parties from leaked information (cf. [47]). ϵ is a DBSCAN parameter indicating the maximal distance between two data records to be considered as neighbors (cf. §3.1).

Our Contributions and Outline

After summarizing related works in §2, we provide the following contributions:

- **Fully private DBSCAN:** We design and implement the first fully private DBSCAN scheme based on secure two-party computation (S2PC) that achieves the same clustering quality as the plaintext algorithm (cf. §5.2). For this, we introduce dedicated protocols for all components of DBSCAN including a *partial parallelization* (cf. §4.3). Our solution can be used for clustering between two parties that want to keep their inputs private as well as for outsourcing scenarios where many data owners secret-share their private input records among two non-colluding servers [38] (cf. §4.1). Due to the usage of generic S2PC, our protocols leak no information beyond the output. They can flexibly be used for arbitrarily partitioned inputs (cf. §4.3.1) and complemented with arbitrary private post-processing of the output (cf. §4.3.5).
- **Privacy-preserving trajectory clustering:** We show that our protocols can be used for the privacy-preserving design of particular instantiations of density-based clustering schemes by introducing the first privacy-preserving trajectory clustering protocol based on the TRACLUS algorithm. TRACLUS is a DBSCAN-based scheme optimized for clustering trajectories, for which we design and implement an approximated distance to improve its secure computation efficiency (cf. §4.4).
- **Open-source implementation and comprehensive empirical evaluation:** We benchmark our implementation[1] and show that our protocols have practical runtimes on public data sets (cf. §5.4). We compare the efficiency of privacy-preserving DBSCAN to state-of-the-art privacy-preserving solutions of the simpler and limited K-means algorithm [35,

[1] https://encrypto.de/code/ppDBSCAN

49] (cf. §5.3). Furthermore, we show that our approximated TRACLUS distance has similar clustering quality as the original distance on public and real-world data sets (cf. §5.2.2).

2 RELATED WORK

In this section, we discuss related work on privacy-preserving K-means and DBSCAN clustering. Moreover, we give a brief overview on privacy-preserving trajectory analysis.

K-means. Privacy-preserving K-means clustering protocols were intensively discussed in the literature. They use different techniques like homomorphic encryption (HE), random blinding, various secure two-party computation (S2PC) and multi-party computation (MPC) techniques, and combinations of the aforementioned.

Most early works assume that all parties have access to a subset of the plaintext input and require that data owners actively participate in the clustering. Private K-means clustering on horizontally partitioned data, i.e., each party holds a subset of the data records, was addressed in [24, 36, 61]. [61, 69] investigate private K-means clustering on vertically partitioned data, i.e., the data owners hold the values of disjoint subsets of attributes of all data records. A flexible mix of both partitioning types is called arbitrary partitioning covered in [14, 33]. Unfortunately, many proposed schemes leak intermediate values such as centroids [24, 36, 46, 61, 69], clusters' sizes [24, 61], merging patterns [32], use incorrect calculations [33], or require active participation of the data owner [46] (which makes them not suitable for the outsourcing scenario). Another research direction leverages differential privacy [8, 64–66] to protect individuals' data, but these works trade accuracy for better performance. Only the following works provide full privacy guarantees while achieving a high accuracy: [14, 35, 49].

Bunn and Ostrovsky [14] create a two-party private K-means protocol using additive homomorphic encryption (AHE) in combination with S2PC for arbitrarily partitioned data. However, the usage of expensive AHE results in impractical runtimes.

Jäschke et al. [35] use fully homomorphic encryption (FHE) for a protocol that enables a data owner to outsource the K-means computation in a privacy-preserving way. To improve runtime, they simplify the required calculation of the original K-means clustering by approximating the centroid determination and distance comparisons, but still the overhead is far from practical for larger data sets.

Mohassel et al. [49] provide an efficient privacy-preserving K-means protocol in the semi-honest security model using Yao's Garbled Circuits. In their work, they improve efficiency by utilizing that K-means requires to calculate the same distance function with one fixed input, namely a point of the input data, to all (repeatedly updated) centroids.

DBSCAN. Although density-based clustering has several advantages as detailed in the beginning of §1, its privacy-preserving realization is barely studied.

Anikin and Gazimov [5] design a privacy-preserving DBSCAN protocol between an arbitrary number of parties with vertically partitioned data in the semi-honest security model. Their scheme uses additive blinding and HE. Besides expensive encryption operations, their protocol requires all parties to have plaintext access to the data records, to be online, and to communicate extensively which makes it not applicable to a privacy-preserving outsourcing scenario. Additionally, one party executes the main part of the computation and obtains the plaintext distances between the input records, cluster assignments, and hence, cluster sizes. The authors do neither provide a concrete instantiation of their protocol nor a performance evaluation.

Similar reasoning applies for [4, 37, 43, 47, 73] who assume vertically/horizontally partitioned data and for [3, 47] with arbitrary data partitioning. They use HE, blinding with random values, and/or partially trusted third parties to protect data privacy. However, they leak information like cluster sizes and neighborhood patterns [3, 37, 43, 47, 58] or distances between input records [4]. Under specific circumstances, the leaked information can be used to concretely identify other parties' input records [43, 47]. In [73], complete data records are revealed when they belong to a cluster. Again, no implementation and performance evaluation were performed in these works.

Rahman et al. [58] introduce a privacy-preserving DBSCAN protocol using key-HE[2] for an outsourcing scenario where an untrusted server conducts the clustering with the help of the data owners. Unfortunately, it leaks information such as the clusters' sizes and neighborhood patterns to the server. Again, no implementation and performance evaluation were performed in that work.

In [52, 72], the authors leverage differential privacy [23] to protect the privacy of individuals' data records. In these works, the data owner performs the clustering, and an untrusted party later requests access to parts of the results. This scenario is not transferable to executing the clustering with inputs from two or more parties. Moreover, the level of noise added to the distance in each parameter dimension depends on the privacy requirements and will inevitably affect the meaningfulness of the data and, hence, the quality of the clustering result.

To summarize, all existing protocols on private density-based clustering assume that the involved parties have access to a fraction of the plaintext input data (meaning that the data owners actively participate in the protocol), use differential privacy (sacrificing accuracy), are not fully private (leaking information), and/or make heavy use of public-key cryptography (making the protocol inefficient and not scalable). Our protocol is the first to be fully precise (i.e., the same clustering quality as with plaintext DBSCAN is achieved), fully private, and efficient for both the data owners and server(s). It is usable in the outsourcing scenario and for two-party computation (cf. §4.1).

Trajectory Analysis. Very few previous studies investigate the problem of private trajectory analysis. Private-Hermes [55] and Hermes++ [41] use anonymisation techniques to generate crafted, realistic fake trajectories to allow users to securely query mobility (trajectory) data sets. Other works such as [2, 27, 53] explore the problem of private ride sharing which consists of finding a match between parts of trajectories. To the best of our knowledge, no previous work has investigated private trajectory clustering. Our privacy-preserving TRACLUS protocol is the first that enables fully privacy-preserving and efficient trajectory clustering between two data owners and in the outsourcing scenario.

[2]Such a HE scheme allows to combine ciphertexts created with different keys to produce an encryption of the sum of the messages decodable with sum of the keys.

3 PRELIMINARIES

In this section, we introduce the original DBSCAN algorithm and its adaptation TRACLUS. We also give the cryptographic building blocks used in our protocols.

3.1 Clustering

3.1.1 DBSCAN. Density-based Spatial Clustering of Applications with Noise (DBSCAN) [20] relies on a density-based neighborhood notion. Elements that are located with many other elements in a dense area form a cluster while data records in a sparse area are marked as noise. This intuition is realized by determining *core points* that have at least minPts other elements in a range of ϵ around them. The range can be quantified with any distance measure. All elements that are located in the neighborhood of a core point p are *directly density-reachable* from p. If an element q is connected via a chain of core points $p_1 p_2 ... p_z$, with $p_z = q$, where each p_{i+1} is directly density-reachable from p_i, then q is called *density-reachable*. All points in the chain have to be core points except from q. A cluster consists of a core point p and all density-reachable elements from p. Elements of a cluster that are not core elements but belong to the neighborhood of a core element are *border elements*. If an element is neither a core element nor a border element it is marked as *noise*.

As mentioned above, DBSCAN has two parameters minPts and ϵ. minPts determines how many data records need to lie in a neighborhood to form a cluster. The larger the value of minPts, the more elements are marked as noise while the clusters get more dominant. Ester et al. [20] suggest minPts = 4 for 2-dimensional data, Sander et al. [62] recommend minPts = 2 · dim where dim denotes the dimensionality of the data, but depending on the application, data set size, and the noisiness of data, the optimal value can vary. ϵ determines the maximum distance between two data records to be considered as neighbors. A large value of ϵ results in large clusters whereas a small value of ϵ marks many elements as *noise*.

DBSCAN can be applied on data of an arbitrary dimension. Its worst case complexity in the cleartext is $O(n^2)$, where n is the number of input elements.

3.1.2 TRACLUS. TRAjectory CLUStering (TRACLUS) [45] optimizes DBSCAN for sequences of multidimensional points (trajectories) and consists of two phases: *partitioning* and *grouping*. In the partitioning phase, the algorithm divides trajectories into subtrajectories that are called *line segments*. Each line segment is represented by two points. The partitioning phase ensures that the output is close to the original trajectory (*preciseness*) and that the number of line segments created is minimal (*conciseness*). Afterwards in the *grouping* (*clustering*) phase, TRACLUS uses DBSCAN with the same parameters (minPts is named minLns for the minimal number of line segments). An optimal ϵ value can be determined with simulated annealing [39]: It is set to the value that minimizes the entropy of the clustering [45].

For the neighborhood determination, TRACLUS uses a special tripartite distance that is illustrated in Fig. 3:

$$dist(L_i, L_j) = w_\perp \cdot dist_\perp + w_{||} \cdot dist_{||} + w_\theta \cdot dist_\theta, \qquad (1)$$

where L_i and L_j are two line segments. The perpendicular distance $dist_\perp$ measures the vertical distance between L_i and L_j. The parallel distance $dist_{||}$ is the horizontal distance between L_i and L_j. The

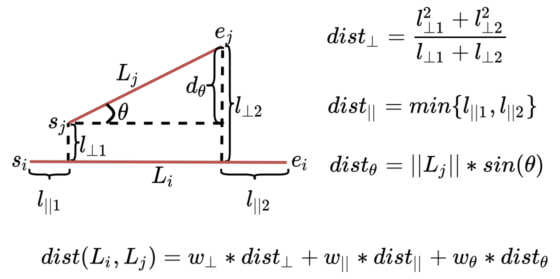

e_j
L_j
d_θ
θ
s_j
$l_{\perp 1}$
$l_{\perp 2}$
s_i
L_i
e_i
$l_{||1}$
$l_{||2}$
$dist_\perp = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}}$
$dist_{||} = min\{l_{||1}, l_{||2}\}$
$dist_\theta = ||L_j|| * sin(\theta)$
$dist(L_i, L_j) = w_\perp * dist_\perp + w_{||} * dist_{||} + w_\theta * dist_\theta$

Figure 3: TRACLUS' tripartite distance. L_i and L_j denote two line segments with start points s_i/s_j and end points e_i/e_j (cf. [45]).

angular distance $dist_\theta$ measures the directional difference between L_i and L_j. The components are weighted with $w_\perp$, $w_{||}$, and w_θ (set to 1 by default) and summed up. See [45] for more details. In §4.4, we give a secure computation-friendly approximation of the tripartite TRACLUS distance.

3.2 Clustering Quality Indices

To evaluate the quality of a clustering method, several quality assessment measures have been proposed. We employ the commonly used Adjusted Rand Index (ARI), Silhouette Coefficient (SC), and Density-Based Clustering Validation (DBCV).

3.2.1 Adjusted Rand Index (ARI). The ARI [30] is an external clustering validation index. External means in this context that the real partitioning, called the ground truth, must be known. It takes two partitionings as an input and evaluates how many input pairs are assigned to the same cluster by both partitionings, how many pairs are assigned to different clusters by both, and how many pairs belong to the same cluster in one partitioning but to different clusters in the other. Additionally, it corrects the count for chance. The ARI is 1 for a perfect clustering. Worse clustering results yield smaller ARIs.

3.2.2 Silhouette Coefficient (SC). As clustering is an unsupervised machine learning technique, a ground truth is normally not available. The most frequently used methodology to evaluate the quality of the clustering result is the silhouette analysis [60]. This methodology analyzes the resulting clusters by evaluating the similarity between the elements within the cluster and the elements of other clusters. The output is called the *silhouette coefficient* (SC) which is close to 1 for a good clustering.

Although SC allows insights about how tightly elements are clustered, it does not take outliers that remain unclustered, called noise, into account. Thus, a penalty is needed to measure the SC with noise as described in [50]. Let n be the data set size and u be the number of not clustered elements. Then, multiplying the SC with $(n - u)/n$ gives the SC with noise penalty, SC_{noise}.

3.2.3 Density-Based Clustering Validation (DBCV). Another method specially designed to assess the quality of density-based clustering algorithms is the DBCV [50]. In contrast to the standard SC, DBCV also takes noise into account. Similar to the other evaluation metrics, a higher DBCV indicates a good clustering quality.

3.3 Secure Two-Party Computation

Secure computation allows two or multiple parties to securely compute on private inputs. In this work, we use secure two-party computation (S2PC) techniques. Typically, S2PC is used for two-party applications, where two parties provide private inputs. Alternatively, it can be extended to an outsourcing scenario [38], where an arbitrary number of input parties secret-share their private input data among two non-colluding servers. Secret-sharing allows to split data into two random-looking values called secret-shares and each of the two servers obtains one of these shares. To recover the data, both servers would have to put their shares together (which by the non-collusion assumption they do not do). The two servers then jointly evaluate a function using S2PC on the shares without being able to learn anything about the inputs or intermediate values.

The non-collusion assumption could be avoided by using homomorphic encryption (HE) [23] but currently available HE schemes have a significant computational overhead as they use expensive cryptographic primitives. Alternatively, using more than two servers with tolerating one corruption yields better efficiency, e.g., [15, 48], but have a larger attack surface as the corruption of any pair of the multiple servers is sufficient to break privacy. Therefore, S2PC is a reasonable trade-off between security and efficiency. Still, our protocols in §4 are general and can easily be extended to more than two parties/servers using hybrid secure multi-party computation frameworks [15, 48].

Security Model. We consider the semi-honest security model where the two non-colluding parties honestly follow the protocol while attempting to collect information about the other party's private inputs. This security model also protects against passive attacks by curious administrators or accidental data leakage. The non-collusion assumption can, e.g., be guaranteed between two competing companies as it is in their interest to not give their customer's data to the competitor and to protect their business secrets. Additionally, in an outsourcing scenario where one or several data owners outsource the expensive computation to (cloud) servers, it is reasonable to assume that the two servers are semi-honest, because cloud providers are strictly regulated and threatened with severe financial damage. Moreover, their reputation is damaged when malicious behavior is detected. Hence, it is in the providers' economic interest to behave semi-honestly. In the outsourcing scenario, our protocols in §4 are even secure against any number of *malicious* data owners who can *arbitrarily* deviate from the protocol [38]. We provide concrete examples how several applications can be realized with S2PC in §4.

ABY. We use the S2PC framework ABY [17] that implements three types of S2PC protocols, namely Yao's Garbled Circuits [74], Boolean GMW [25], and Arithmetic sharing, which is a generalization of GMW, including state-of-the-art optimizations. Moreover, it enables flexible conversions between the three sharing types. ABY supports Single Instruction Multiple Data (SIMD) operations that efficiently apply the same functionality on multiple inputs in parallel. SIMD operations reduce memory usage and evaluation time. We use Yao's Garbled Circuits (i.e., Yao sharing) and Arithmetic sharing in this work (cf. §B for details).

4 PRIVACY-PRESERVING DENSITY-BASED CLUSTERING

In this section, we present our protocols for privacy-preserving DBSCAN (ppDBSCAN) clustering, which for the first time provides scalability with respect to data set size, the number of clusters, and input records' dimension as well as full privacy while maintaining the clustering quality of the plaintext algorithm. We extend our protocol to the private clustering of trajectories using the TRACLUS algorithm (ppTRACLUS).

4.1 Application Scenarios

As already indicated in §3.3, our ppDBSCAN and ppTRACLUS can be used in two application scenarios:

- **Two-party Computation (2PC)**: Here, two data owners privately perform the clustering on their vertically or horizontally partitioned data. In such a classical secure two-party computation, each of them runs one semi-honest party itself. The data owners could, for example, be two banks who jointly investigate credit card frauds.
- **Outsourcing**: Here, one or multiple data owners outsource their data as random-looking secret-shares (cf. §3.3) to two non-colluding semi-honest parties (e.g., cloud servers) that perform the private clustering via S2PC. The data owners want to hide their sensitive data from any other party. As explained in §3.3, it is in the cloud providers' interest to behave semi-honestly and to not collude. The data owners can even be malicious and arbitrarily deviate from the protocol [38]. Considering the application examples in §1, the data owners can be, for example, multiple hospitals that join forces to improve medical diagnosis or a telecommunication provider that lawfully wants to monetize location information of customers by selling privacy-preserving analytics to the travel industry. To analyze people's movements in a pandemic, a governmental institution and a non-profit organization like the EFF[3] could operate the two non-colluding servers.

In the following, a *party* denotes one of the two non-colluding and semi-honest parties that perform the privacy-preserving clustering.

4.2 Notation

We use the following notation in the remainder of this work: A multiplexer gate (MUX) is denoted by $a?b{:}c$ which corresponds to if a then b; else c. An AND gate is denoted by $\wedge$ and an OR gate by $\vee$.

4.3 Privacy-preserving DBSCAN

In this section, we present our secure two-party computation (S2PC) protocol for fully privacy-preserving DBSCAN including a partial parallelization of the clustering. A discussion on its security properties is given in §C.

4.3.1 Flexible Inputs. Initially, the data owners use Arithmetic sharing (cf. §3.3) to secret-share their data among each of the two parties. Each of the two parties collects the secret-shares of the inputs it receives from all data owners in a vector X that is input

[3]https://www.eff.org

to the clustering. Additionally, the two processing parties can also be provided with secret-shares of DBSCAN's parameters minPts and ϵ (cf. §3.1) such that they do not know them in the clear.

Note that the usage of Arithmetic sharing for the input data that shall be clustered enables an *arbitrary* number of data owners to easily join the clustering as no encryption key is needed. Moreover, our ppDBSCAN can support *any* data splitting, i.e., horizontally or vertically distributed data or an arbitrary mix of both, because thanks to the Arithmetic sharing the data owners can simply secret-share the attributes of the data records they are holding and send these shares to the two parties who can sort them according to the respective data records for the clustering.[4]

After receiving all shares from the data owners, the clustering starts. DBSCAN can be split into two main building blocks, the *distance calculation* and the *clustering* process, for which we design novel S2PC protocols enabling a partial parallelization.

Listing 1: Privacy-preserving Squared Euclidean Distance (SED) calculation.

```
1  Input: vector X // n-dim. vector with input data
2         $\epsilon^2$        // sq. threshold
3  Output: dist     // n-dim. vector with SIMD -values:
4                   // dist[l]=(SED(X[k],X[l])< $\epsilon^2$), $0 \leq k < n$ */
5
6  sharedElements = combineToSIMD(X)
7  Vector dist;
8  for(i=0; i < n; i++):
9    simdElement = [X[i]]*n // n copies of X[i]
10   sed = 0
11   for(j=0; j < dim; j++):
12     temp = sharedElements[j] - simdElement[j] // Arithm
13     sed += temp * temp
14   dist[i] = sed < $\epsilon^2$ // Yao
15 return dist
```

4.3.2 Parallelized Distance Calculation. In DBSCAN, pairwise distances between all input records are calculated to determine neighborhoods. In general, any distance measure can be used. Ester et al. [20] originally use the Euclidean distance (ED), but for more efficient evaluation we use the squared Euclidean distance (SED) which is common for S2PC, e.g., privacy-preserving face recognition [18]. To determine if two elements are neighbors, the ED of two input values x and y has to be smaller than ϵ which is equivalent to $SED(x, y) < \epsilon^2$.

List. 1 presents our private SED protocol. To massively parallelize the computation, each party combines its secret-share of the jth coordinates of each data record in X with *combineToSIMD(..)* in one SIMD arithmetic share (cf. §3.3) and collects these SIMD-shares in a j-dimensional vector *sharedElements*. For every execution of the *for* loop in Line 8, a j-dimensional vector *simdElement* is created by each party where the jth entry contains n times its secret-share of the jth coordinate of the ith input, where n denotes the data set size. In Lines 11-13, the SED is jointly calculated between the two parties. Then, they jointly compare the result to the threshold ϵ^2 and only the comparison result (secret-shared between the two parties) is saved in the vector *dist* to reduce the storage requirement to 1 bit per input record pair for each party. Given the symmetry of the distance measure, this approach causes an overhead as it calculates n^2 instead of $\approx 0.5n^2$ distances. But this is compensated by enabling parallel distance computation and a partial parallelization of the clustering which substantially reduces memory consumption and improves runtime. For multiplications and additions, Arithmetic sharing is most efficient (addition can even be done for free), and we convert the resulting shared SED to Yao sharing for efficient comparison [17].

We point out that further optimizations for distance calculations from [34, 49] can improve the runtime and/or communication costs of our protocol. However, as our experiments in §5.4 show, computing SEDs is only a small fraction of the total complexity, so we did not implement further optimizations.

4.3.3 Parameter Estimation. As described in §3.1, DBSCAN has two parameters: ϵ (now ϵ^2 because of the SED) determines the maximal distance between two input records to consider them as neighbors, and minPts is the minimal cluster size.

The minimal cluster size minPts is strongly dependent on the respective use case. For example, if for the containment of COVID-19 contact restrictions have been put into place and meetings with only less than 10 people are allowed, minPts could be set to 10 to check if and how many "illegal" meetings happened. We recommend to choose minPts based on the use case and its privacy requirements as well as the data set size. Generally, it is advantageous to aim for larger clusters (i.e., to choose a sufficiently large minimal cluster size minPts) as they generalize better and leak less information. For our DBSCAN experiments in §5, we follow the recommendation by [20] and set minPts $= \frac{n}{100}$ (resp. minPts $= 4$ for the smaller data set), where n is the data set size.

Similarly, for some use cases the value for ϵ^2 might already be given by the application itself: When investigating the movements of people for the containment of COVID-19, ϵ^2 can be set to 2 meters similarly as it is done in contact tracing apps [67]. Alternatively, ϵ^2 can be estimated by plotting a sorted k-distance graph as suggested in [20]. In an outsourcing scenario with a single data owner, the data owner can determine the value of ϵ^2 locally. When several data owners provide input data and the data is expected to be approximately independent and identically distributed, it can be sufficient that one data owner DO_i plots the graph with his data setting $k = \text{minPts}/\frac{n_i}{n}$, where n_i denotes the size of DO_i's input set. The proportion $\frac{n_i}{n}$ and n itself may also be approximated. Otherwise, the data owners can also agree on a value for ϵ^2 after each data owner analyzed its data locally by a secure aggregation of their local results.[5] Furthermore, the calculation of a k-distance graph and determination of ϵ^2 can also be realized with a secure computation ahead of the clustering. However, such a secure computation protocol for calculating ϵ^2 is not the focus of our work and we leave it open for future work. Some works have already investigated private sorting [28, 34] and computing of the k-th nearest neighbor [16, 34, 57].

The minimal possible k is 2 and the minimal meaningful value for minPts is 3. As noted in the beginning of §4.3, both parameters can be secret-shared and no information is leaked.

[4]Even the data set size can be hidden from the two processing parties by using dummy data records, but one has to ensure that the added data records do not change the clustering result. One option can be to duplicate data records in combination with increasing the value of minPts. Also the number of parameters per data record can be hidden by adding dummy parameters that are set to the same value for all input records such that they do not affect the clustering result.

[5]Secure aggregation protocols have been thoroughly investigated in the context of smart metering, e.g., [19, 22, 44].

Listing 3: Privacy-preserving DBSCAN (ppDBSCAN) with partial parallelization.

```
1  Input: vector sharedElements /* n-dim. vector with
       SharedInputRecords (cf. Listing 2)*/
2  Output: sharedElements
3
4  // clustering in Yao
5  currentClusterId = 0
6  for each elem in sharedElements:
7    elem.clusterId = elem.notProcessed?
           (elem.validNeighborhoodSize?currentClusterId:0):
9          elem.clusterId
10   isCoreElement = elem.validNeighborhoodSize∧
11         elem.notProcessed
12   elem.isNoise = elem.notProcessed?
13         (!elem.validNeigborhoodSize):elem.isNoise
14   elem.notProcessed = 0
15
16   sIsCoreElement = [isCoreElement]*n        // n copies
17   sCurrentClusterId = [currentClusterId]*n // n copies
18
19   for(i=0; i < maxIterations; i++):
20     /* create SIMD values with the value of the
           respective attribute of all sharedElements */
21     sClusterIds, sIsNoise, sNotProcessed,
           sValidNeighborhoodSize = combineSharesToSIMD(
           sharedElements)
22
23     sValidNeighbor = ((sIsNoise∨sNotProcessed)∧
           sIsCoreElement)∧elem.neighbors
24     sClusterIds = sValidNeighbor?sCurrentClusterId:
           sClusterIds
25     sNotProcessed = sValidNeighbor?0:sNotProcessed
26     sIsNoise = sValidNeighbor?0:sIsNoise
27     // check if neighborhood size and if valid neighbor
28     sNeighborsAreCoreElement = sValidNeighbor?
           sValidNeighborhoodSize:0
29
30     for (k=0; k < n; k++):
31       sUpdateResultDistanceComp =
32       sNeighborsAreCoreElement∧sharedElements[k].
           neighbors
33       elem.neighbors[k] = ORTREE(
           sUpdateResultDistanceComp)?1:elem.neighbors[k]
34
35     // split simd values and update sharedElements
36     updateSharedElements({sIsNoise, sNotProcessed},
           sharedElements)
37   updateSharedElements({sClusterIds}, sharedElements)
38   currentClusterId = currentClusterId+isCoreElement //
         Arithm
39 return sharedElements
```

Listing 2: Shared input element.

```
class SharedInputRecord(minPts, // (minimal neighborhood
    size
    splittedD): /* (n-dim. vector with subset of splitted
        dist output by Listing 1)*/
  clusterId = 0
  isNoise = 0
  notProcessed = 1
  Vector neighbors = splittedD
  validNeighborhoodSize = minPts < hamming_weight(
      splittedD)
```

4.3.4 *Partially Parallelized & Private Clustering.*

To make DBSCAN fully private, it is not sufficient to have a private distance calculation, but also all intermediate information such as the clusters' sizes, assignment patterns, and number of clusters must not be leaked as these can contain sensitive information.

The plaintext DBSCAN clustering is given in §A. We significantly reduce the overhead of the neighborhood queries for every record by calculating all distances in a parallelized fashion before starting the clustering instead of doing it several times for every single record like in the plaintext. For every input record, each party creates an object of *SharedInputRecord* (cf. List. 2) which contains its secret-shares of the object's attributes' values. The secret-shares of the results of the comparison of the record's distances to all other records with ϵ^2 are stored in *neighbors*. The combined secret-shares of both parties of *validNeighborhoodSize* denote if an input has more than minPts neighbors, i.e., whether it is a core element (cf. §3.1). It follows that minPts has to be one less than in the plaintext DBSCAN to obtain the same clustering result.

Each party collects all objects of *SharedInputRecord* in a vector *sharedElements* and inputs it into our private clustering protocol in List. 3. In Lines 7 to 14 in List. 3, the parties jointly check if the data record *elem* was assigned to a cluster before and, if this is not the case, they check if *elem* is a core element such that it creates a new cluster or it is marked as noise. If a new cluster is created, the unclustered neighbors of *elem* are also added to this cluster (Lines 23-26 in List. 3). If the neighbors themselves are core elements, their neighbors also have to be analyzed and added to the cluster (Lines 28-36 in List. 3) as they are density-reachable (cf. §3.1). In plaintext DBSCAN, this is realized with a queue which leaks neighborhood patterns (cf. Lines 19-24 in List. 4). Therefore, for a privacy-preserving realization, all elements always have to be checked instead to obliviously expand a cluster. This allows us to parallelize these steps shown in Lines 19 to 36 in List. 3 with SIMD operations. However, as *elem*'s neighbors are changing, one has to repeat the loop starting in Line 19 in List. 3 maximally n times to accommodate the neighbors of neighbors that are core elements through the update in Line 33 in List. 3. This would increase the complexity of the protocol to $O(n^3)$ to make it fully oblivious. But depending on the data distribution few *maxIterations* are already sufficient. This yields a complexity of $O(maxIterations \cdot n^2)$. In our tests on the data sets used in §5, setting $maxIterations = 4$ detects all clusters correctly. Generally, many iterations are only needed if clusters consist of core elements that are chained in a row; thus, simply speaking, if clusters are particularly elongated. If the data owners cannot estimate the needed *maxIterations* in advance, a privacy-preserving check can be added that tests if the vector *elem.neighbors* in Line 33 in List. 3 changed in the last iteration (or after a specific number of iterations). This leaks one bit of information to the two parties to inform them if they have to continue further iterations. This additional information leaks only vaguely how the input data is distributed which can be acceptable for applications as a trade-off for better efficiency. The function $combineSharesToSIMD(..)$ takes a vector with n *SharedInputRecord*s as input and combines their $clusterId$-, $isNoise$-, $notProcessed$-, and $validNeigborhoodSize$-values to SIMD shares each containing n-values. $updateSharedElements(..)$ does the opposite, it takes SIMD shares as input, splits them, and updates the corresponding *SharedInputRecord*'s objects.

All bit-operations of our ppDBSCAN protocol in List. 3 are done in Yao sharing and the addition in Line 38 is done with Arithmetic sharing to optimize efficiency [17]. Furthermore, note that List. 3 is independent of the number of clusters and the dimensionality of the input. Only the distance calculation in §4.3.2 is influenced by the dimensionality while the number of clusters has no effect on the performance of ppDBSCAN. Its clustering's complexity solely depends on the input data set's size n.

4.3.5 Flexible Output and Post-Processing. Note that our protocol is applicable for both the outsourcing scenario where one or more (possibly malicious) data owners outsource computation to two non-colluding semi-honest (cloud) servers as well as for two-party applications (cf. §3.3 and §4.1). In both scenarios, the output can be provided to a third party, one or multiple data owners, or one of the servers depending on the use case.

Furthermore, the output of a clustering can flexibly be adapted to the application without leaking intermediate results. For example, the clustering can return a cluster label for each data record. Alternatively or additionally, it can return the average (i.e., centroid) or medoid of each cluster, or the number of clusters and respective cluster sizes.

Our secure two-party computation (S2PC) protocols for ppDBSCAN allow to realize each of these options as well as any further post-processing in a secure and efficient way by simply defining what is sent back to the respective parties that shall receive the output and optionally securely evaluating a post-processing circuit via S2PC.

4.4 Privacy-preserving TRACLUS

In this section, we provide a S2PC protocol for privacy-preserving trajectory clustering. For trajectory clustering with our privacy-preserving clustering protocol based on TRACLUS (cf. §3.1.2), we assume that the data owners hold a horizontally split data set, i.e., each data owner holds full trajectories.[6] They locally run the TRACLUS partitioning phase and outsource the clustering phase by providing arithmetic secret-shares (cf. §B) of the line segments to the two processing parties.

Simplified and Approximated Distance Measure. As described in §3.1.2, the computation of the TRACLUS distance involves complex operations like sine computations which are relatively expensive in S2PC (cf. §3.1.2). In order to ensure data privacy and execute the clustering phase in an efficient manner, we simplify this distance calculation using an approximation. Afterwards, ppDBSCAN — as presented in List. 3 — is executed.

As the Euclidean Distance (ED) is the most common measure used for trajectory clustering [12], we propose to replace TRACLUS' original distance with a combination of EDs. The perpendicular, parallel, and angular distances are still partially taken into account when using the EDs between each pair of points of the two line segments. As already explained in §4.3.2, ED can be replaced by the Squared Euclidean Distance (SED). Therefore, given line segment L_i with start point $s_i = (x_{si}, y_{si})$ and end point $e_i = (x_{ei}, y_{ei})$ and similar for line segment L_j, our approximated distance measure realized with arithmetic sharing for privacy-preserving TRACLUS (ppTRACLUS) is defined as:

$$\begin{aligned} d_{pptrac}(L_i, L_j) = &SED(s_i, s_j) + SED(s_i, e_j) + \\ &SED(e_i, s_j) + SED(e_i, e_j). \end{aligned}$$

To summarize, the two processing parties first jointly compute d_{pptrac} and compared it then to ϵ' in order to check whether two line segments are neighbors.[7]

5 EXPERIMENTAL EVALUATION

In this section, we present the experimental results of our ppDBSCAN and ppTRACLUS. We run the experiments with four public data sets and one private real-world data set to show clustering quality and to benchmark runtime and communication costs. Additionally, we compare to previous work on private K-means to show the practicability of our ppDBSCAN.

5.1 Experimental Setup

Server Configuration. We implement our protocols using the ABY framework [17] (cf. §3.3) which is written in C++ and uses 64-bit precision. The experiments are performed on two separate servers each equipped with Intel Core i9-7960X CPUs with 2.8 GHz and 128 GB RAM. As in an outsourcing scenario, the two servers are well-connected over a 10 Gbit/s LAN with 0.2 ms RTT.

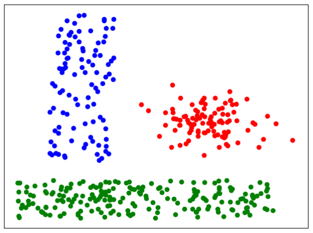
Scenario. As detailed in §4.1, ppDBSCAN can be run in an outsourcing or a 2PC scenario. Moreover, the data to be clustered can be arbitrarily split among the data owners for ppDBSCAN. For ppTRACLUS, we require a horizontal data splitting to enable each data owner to locally execute the partitioning phase. In our benchmarks, we secret-share and outsource all data records to the two non-colluding servers to assess the efficiency and quality of our ppDBSCAN and ppTRACLUS. This models an outsourcing scenario with a single data owner, but we stress that the clustering (and hence its efficiency and quality) is *independent* of the number of data owner(s) and the data splitting between the data owner(s) in the outsourcing scenario as well as of the data splitting between the two parties in the 2PC scenario. It follows that our experimental results are directly transferable to other scenarios and more data owners as long as the data set size, the number of parameters, and the number of clusters is equal to our reported setting.

Data Sets. For the evaluation of our private clustering protocols, we use five data sets from different sources. Two are chosen based on previous work on private K-means [35, 49] to evaluate and compare the efficiency and practicality of our ppDBSCAN. Additionally, we evaluate the clustering quality of our simplification of the original TRACLUS distance with a real world data set that contains location data extracted from mobile phones and two public data sets, namely the Hurricane and the Deer data sets used in the original TRACLUS paper [45].

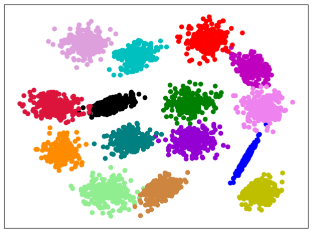
- **Lsun:** The Lsun data set [68] in Fig. 4a was used in [35, 49] for evaluating private K-means protocols. It contains 400 2-dimensional data points. Its ground truth contains 3 clusters with different variances and cluster distances. The clusters have 200, 100, and 100 elements.

[6]Considering, e.g., the use case of collecting human movements trajectories for analysis purposes related to the containment of COVID-19, this trajectory data is typically collected by telecommunication providers such that assuming horizontally split data is plausible [9].

[7]We indicate the ϵ of ppTRACLUS with ϵ' to unambiguously differentiate it from the ϵ^2 used in ppDBSCAN.



(a) Lsun



(b) S1

Figure 4: Ground truth of data sets Lsun and S1.

Table 1: Clustering quality evaluation.

Data Set	K-means	DBSCAN
Lsun	0.4386	1.0
S1	0.9254	0.9757

- **S1:** S1 [21] in Fig. 4b is a 2-dimensional synthetic data set with 5 000 samples that can be split into 15 spherical Gaussian clusters with between 300 and 350 elements. The clusters have an overlap of 9%. The set was used in [49, 66] to evaluate private K-mean protocols.
- **Travel:** This "synthetic" data set (not related to real individuals) is a 2-dimensional trajectory data corresponding to location of people's mobile phones. It has been created and provided by Orange S.A.[8], a major telecom provider, based on anonymized indicators of real trajectories. It consists of 40 000 line segments.
- **Hurricane:** This data set [45] has 2-dimensional track data of Atlantic hurricanes from 1950 to 2006. It has 608 trajectories with 18 343 line segments.
- **Deer:** This data set [45] corresponds to 2-dimensional movements of deers in 1995. There exist 32 trajectories which correspond to 20 033 line segments.

Encoding. The Lsun data set consists of rational numbers. Similarly as in previous work [35], we scaled the data to an integer representation with a factor of 10^6 to use efficient S2PC protocols on integers [17]. The S1, Deer, Hurricane, and Travel data sets were already represented by integers.

5.2 Clustering Quality

In the following, we compare the clustering quality of the output of DBSCAN (which is exactly realized by ppDSCAN) to the results of the well-known K-means algorithm on the data sets Lsun and S1. Afterwards, we demonstrate that our simplified TRACLUS distance (cf. §4.4) provides a better clustering quality than the original tripartite distance of TRACLUS (cf. §3.1.2) on our tested data sets.

5.2.1 Comparison between DBSCAN and K-means. As clustering is an unsupervised ML technique, it is normally not possible to calculate an *accuracy*, meaning the proportion of correct predictions/ classifications in supervised machine learning. However, Lsun and S1 are artificial data sets, created to benchmark machine learning algorithms, so their ground truths are known. For this reason, we are able to evaluate the clustering quality of the outputs of K-means and DBSCAN with the Adjusted Rand Index (ARI, cf. §3.2) which is a widely used *external* clustering quality measure [6, 70].

For a fair comparison, we provide both clustering algorithms with optimal values for their parameters. For K-means, centroids are initialized at random, and we provide the right amount of clusters ($K = 3/15$). For DBSCAN, we choose the parameters as described in §4.3.3: $\epsilon = 2 \times 10^{11}$ and $\mathsf{minPts} = 4$ with Lsun, $\epsilon = 2.25 \times 10^{9}$ and $\mathsf{minPts} = 50$ with S1.

Note that we use the Squared Euclidean Distance for DBSCAN such that the output of DBSCAN is equal to the output of ppDBSCAN. Thus, the insights derived from the clustering quality evaluation in this section are directly transferable to the clustering quality of ppDSBCAN.

The results shown in Tab. 1 are averaged across 10 experiments. DBSCAN finds a better partitioning than K-means for both data sets. Especially for Lsun, DBSCAN performs significantly better than K-means. This is easily explainable by the data distribution of both data sets (cf. Fig. 4a and Fig. 4b). K-means cannot correctly split Lsun, because it only detects convex clusters which works relatively well for the mostly round shaped clusters in S1, but not for the two rectangular shaped ones in Lsun.

5.2.2 Validation of Simplified and Approximated TRACLUS Distance. We compare the clustering quality of ppTRACLUS with the original TRACLUS. As no ground truth is known for the trajectory data sets Hurricane, Deer, and Travel, we rely for this on well established *internal* clustering quality indices [6, 42]: the Silhouette Coefficient (SC) [60], SC_{noise}, and Density-Based Clustering Validation (DBCV) [42] (cf. §3.2). We conduct experiments with various values for ϵ' and minLns for the Hurricane data set [45], the Deer data set [45], and the Travel data set. As in [45], the optimal values for ϵ' and minLns are computed with simulated annealing. The ϵ' values differ for TRACLUS and ppTRACLUS because of the different distance measures. Note that all ϵ' for an Experiment $i \in \{1, 2\}$ result in the same entropy level in simulated annealing.

Tab. 2 contains the results for the Hurricane data set. In Experiment 1, TRACLUS outputs one cluster less than ppTRACLUS. Moreover, the number of elements marked as noise and the number of clusters are larger with ppTRACLUS than with the original plaintext TRACLUS. Nevertheless, we observe that ppTRACLUS outputs better results with respect to SC, SC_{noise}, and DBCV than the original TRACLUS. Results for the Deer data are given in §D.1.

Tab. 3 contains the results of Experiment 1 of ppTRACLUS and TRACLUS for the Travel data set. Tab. 6 in §D.1 contains the results of Experiment 2 of ppTRACLUS and TRACLUS for the Travel data set. We notice the same behavior with respect to the quality evaluation metrics as for the Hurricane and Deer data sets shown in Tab. 2 and Tab. 5 in §D.1. In Experiment 1, the number of input records marked as noise by ppTRACLUS is larger than the number of elements marked as noise by the original plaintext TRACLUS. However, when fine-tuning the values of ϵ' and minLns, it is possible to decrease the number of elements marked as noise (cf. Tab. 6 in §D.1 and §5.4 in [45]). Both algorithms output relatively few clusters while this particular data set (illustrating peoples' travel

[8] https://www.orange.fr

Table 2: Clustering quality assessment for TRACLUS and ppTRACLUS on the Hurricane data set. For all scores larger values are better (best marked in bold).

Data Set	Score	Experiment 1		Experiment 2	
		TRACLUS	ppTRACLUS	TRACLUS	ppTRACLUS
Hurricane	(ϵ', minLns)	(24, 5)	(5000, 5)	(4, 5)	(2250, 5)
	# of Clusters	2	3	11	13
	Noise	129	150	597	645
	SC	0.27	**0.87**	0.44	0.79
	SC$_{noise}$	0.27	**0.86**	0.42	0.76
	DBCV	0.72	**0.97**	0.64	0.76

Table 3: Results of Experiment 1 of the clustering quality assessment for TRACLUS, ppTRACLUS, and ppTRACLUS' on the Travel data set. The best results are marked in bold.

	Experiment 1		
	TRACLUS	ppTRACLUS	ppTRACLUS'
(ϵ', minLns)	(4200, 3)	(450×10^6, 3)	(450×10^6, 3)
# of Clusters	1	2	48
Noise	796	13092	13506
SC	N/A	0.83	**0.98**
SC$_{noise}$	N/A	0.56	**0.65**
DBCV	N/A	**0.67**	0.37

patterns) may need more precision (i.e., more clusters). One illustrative example is the identification of typical routes between two locations A and B. In this example and for this particular data set, ppTRACLUS groups all line segments in one cluster because of its expansion method (cf. Line 19 of List. 3). We propose to reduce *maxIterations* to 1 in order to take only the first-level neighbors into account and to obtain a larger variety of clusters (containing trajectories that reach B by passing through different locations $L_i, 0 \leq i \leq maxLoc$) and call this adaption ppTRACLUS'. We run the same two experiments for ppTRACLUS'. Our results show that the number of clusters increases with good results for SC, SC$_{noise}$, and DBCV.

To summarize, our simplified and approximated distance measure d_{pptrac} tends to create a greater number of clusters (resulting in smaller clusters in average) and marks more elements as outliers. Nevertheless, our quality evaluation with well-established clustering quality indices surprisingly gives even better results for the three analyzed data sets showing that its performance is comparable to the original tripartite distance measure of TRACLUS.

Intuition for better Clustering Quality of ppTRACLUS. Intuitively, our approximated distance d_{pptrac} summarizes all possible location-based differences between two line segments while the original tripartite distance explicitly focuses on horizontal, vertical, and angular distance. Hence, the original distance might unnecessarily amplify differences of the three different distance types although those might be partially equalized when considering the combined effect as done by d_{pptrac}. It follows that our approximation might generalize better than the original which could be the reason for the better clustering results. To conclude, ppTRACLUS with our d_{pptrac} offers efficient privacy protection with high clustering quality.

5.3 Runtime Comparison and Evaluation

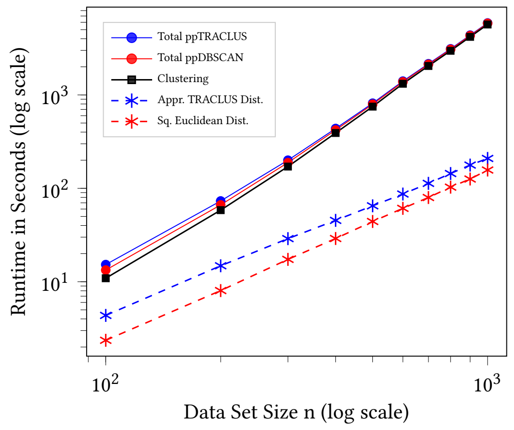
In this section, we compare the runtime of our ppDBSCAN to two state-of-the-art fully privacy-preserving clustering schemes presented in [35, 49]. These two works focus on the K-means algorithm which is significantly less powerful than DBSCAN and, thus, often yields worse clustering results than DBSCAN (cf. §1). We do on purpose not experimentally compare to the previous works on private DBSCAN as all of these works leak intermediate information whereas our protocol does not (cf. §2) . Moreover, none of these works was implemented and experimentally evaluated.

Tab. 4 contains the runtimes of two state-of-the-art private K-means protocols [35, 49] with the Lsun data set and of [49] with the S1 data set. Both works use a better network than we do: [35] uses a single Intel i7-3770 with 3.4 GHz and 20 GB RAM. [49] ran their experiments locally on an Intel Core i7, 2.6 GHz with 12 GB RAM over a simulated LAN with 10 Gbit/s and 0.02 ms RTT. For both works, we present the best achieved runtimes, which is in [49] an exact calculation with 15/30 iterations. [35] simplifies the original K-means algorithm to get rid of divisions with encrypted denominators that are originally needed to update the centroids as they are expensive to realize with HE. Their fastest approximation needs 15.47 h for one iteration but it takes 40 iterations until convergence and sacrifices some accuracy for this speed-up. Our runtime for Lsun is averaged across 10 experiments. Because of time constraints, we approximate the overall runtime of our protocol for the S1 data set by multiplying the average runtime of one iteration by the data set size and adding the average distance calculation time. We use the same DBSCAN parameters as in §5.2.

Table 4: Runtime comparison of private clustering schemes.

Algorithm	Privacy-preserving K-means		ppDBSCAN
Data Set	Jäschke et al. [35]	Mohassel et al. [49]	This work
Lsun	25.79 days	$22.21s$	$420.72s$
S1	-	$1,472.60s$	$620,912.70s$

Tab. 4 shows that our ppDBSCAN is about 19x (for Lsun) and 422x (for S1) slower than the private K-means protocol of [49], but more than 5 000x faster than the HE-based private K-means protocol of [35] on the small data set Lsun. While in K-means optimizations like batching [49] reduce runtime, making DBSCAN fully private



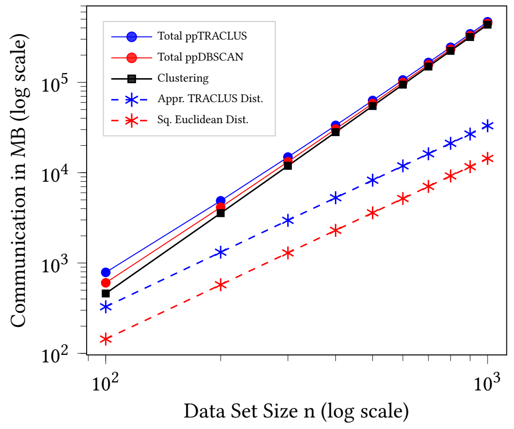


Figure 5: Runtimes and communication of our ppDBSCAN and ppTRACLUS.

adds additional computation compared to the plaintext algorithm. Again, we want to emphasize that DBSCAN is more complex than K-means but also more powerful since it can detect arbitrarily shaped clusters, automatically determine the required number of clusters, and handle noise, which results in a higher clustering quality.

5.4 Scalability

Although DBSCAN has an inherent worst case complexity of $O(n^2)$, it is one of the most used clustering algorithms because of its favorable properties. Making it private inherently implies additional overhead. Still, our protocol is practical as shown in Fig. 5 (exact numbers are provided in §D). The complexities for computing the two distances scale quadratically in the data set size, while the clustering has a low cubic complexity.

The squared Euclidean distance (SED) is applied on 2-dimensional data, d_{pptrac} on 4-dimensional line segments. *maxIterations* is set to 4. As discussed in §4.3.4, an increase of the inputs' dimension will only affect the runtime and communication of the distance calculation, but not of the clustering process. The distance calculation scales linearly in the dimensionality of the input records. Moreover, a larger number of clusters does not change our clustering and therefore also not the clustering's costs. To summarize, the private clustering component of ppDBSCAN is *independent* of the number of clusters and the data dimensionality. In contrast, private K-means [35, 49] requires to newly calculate the distances to the centroids in every iteration, such that its efficiency is heavily affected by an (1) increased input dimension, (2) a higher number of clusters (i.e., more centroids), and (3) it leaks the number of clusters K by design.

6 CONCLUSION

In this work, we presented the first fully private DBSCAN based on secure two-party computation. We designed efficient protocols for ppDBSCAN and introduced a partial parallelization for the clustering. Furthermore, we showed that our protocols can be extended to other density-based clustering algorithms by introducing the first private trajectory clustering which has interesting real-world applications for financial time series forecasts or analyzing people's movements in a pandemic. We designed a S2PC-friendly approximated distance measure for trajectories and evaluated its quality showing that it can even offer a better clustering quality than the original TRACLUS distance. Finally, we demonstrated ppDBSCAN's efficiency in terms of runtime and communication with benchmarks on real-world and public data sets and compared its overhead to state-of-the-art private K-means [35, 49] whose limitations we overcome.

ACKNOWLEDGMENTS

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement no. 850990 PSOTI) and from the PAPAYA project funded by the European Union's Horizon 2020 research and innovation program (grant agreement no. 786767). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/ 251805230, and by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within ATHENE.

REFERENCES

[1] M. Ahmed, A. N. Mahmood, and Md. R. Islam. 2016. A Survey of Anomaly Detection Techniques in Financial Domain. In *Future Generation Computer Systems*.
[2] U. M. Aïvodji, K. Huguenin, M. Huguer, and M. Killijian. 2018. Sride: A Privacy-Preserving Ridesharing System. In *WISEC*. ACM.
[3] N. Almutairi, F. Coenen, and K. Dures. 2018. Secure Third Party Data Clustering Using Φ Data: Multi-User Order Preserving Encryption and Super Secure Chain Distance Matrices. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*.
[4] A. Amirbekyan and V. Estivill-Castro. 2006. Privacy Preserving DBSCAN for Vertically Partitioned Data. In *Intelligence and Security Informatics*. Springer.
[5] I. V. Anikin and R. M. Gazimov. 2017. Privacy Preserving DBSCAN Clustering Algorithm for Vertically Partitioned Data in Distributed Systems. In *International Siberian Conference on Control and Communications*. IEEE.
[6] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. PéRez, and I. Perona. 2013. An Extensive Comparative Study of Cluster Validity Indices. *Pattern Recognition* (2013).
[7] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. 2013. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *CCS*. ACM.
[8] M.-F. Balcan, T. Dick, Y. Liang, W. Mou, and H. Zhang. 2017. Differentially Private Clustering in High-Dimensional Euclidean Spaces. In *International Conference on Machine Learning (ICML)*. PMLR.
[9] A. Bampoulidis, A. Bruni, L. Helminger, D. Kales, C. Rechberger, and R. Walch. 2020. Privately Connecting Mobility to Infectious Diseases via Applied Cryptography. https://eprint.iacr.org/2020/522.
[10] D. Beaver. 1991. Efficient Multiparty Protocols Using Circuit Randomization. In *CRYPTO*. Springer.

[11] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. 2013. Efficient Garbling from a Fixed-Key Blockcipher. In *S&P*. IEEE.

[12] P. Besse, B. Guillouet, J.-M. Loubes, and F. Royer. 2016. Review & Perspective for Distance Based Clustering of Vehicle Trajectories. In *Transactions on Intelligent Transportation Systems*. IEEE.

[13] Beyza Bozdemir, Sébastien Canard, Orhan Ermis, Helen Möllering, Melek Önen, and Thomas Schneider. 2021. Privacy-preserving Density-based Clustering. In *ASIACCS*.

[14] P. Bunn and R. Ostrovsky. 2007. Secure Two-Party K-means Clustering. In *CCS*. ACM.

[15] H. Chaudhari, R. Rachuri, and A. Suresh. 2020. Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. In *NDSS*. The Internet Society.

[16] H. Chen, I. Chillotti, Y. Dong, O. Poburinnaya, I. Razenshteyn, and M. S. Riazi. 2020. SANNS: Scaling Up Secure Approximate k-Nearest Neighbors Search. In *USENIX Security*. USENIX.

[17] D. Demmler, T. Schneider, and M. Zohner. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*. The Internet Society.

[18] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. 2009. Privacy-Preserving Face Recognition. In *PoPETS*. Springer.

[19] Z. Erkin, J. R. Troncoso-pastoriza, R. L. Lagendijk, and F. Perez-Gonzalez. 2013. Privacy-preserving Data Aggregation in Smart Metering Systems: An Overview. In *IEEE Signal Processing Magazine*.

[20] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. 1996. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM.

[21] P. Fränti and S. Sieranoja. 2018. K-means Properties on Six Clustering Benchmark Datasets. In *Applied Intelligence*. Springer.

[22] F. D. Garcia and B. Jacobs. 2010. Privacy-friendly Energy-metering via Homomorphic Encryption. In *International Workshop on Security and Trust Management*. Springer.

[23] C. Gentry. 2009. *A fully Homomorphic Encryption Scheme*. Stanford University.

[24] Z. Gheid and Y. Challal. 2016. Efficient and Privacy-Preserving K-means Clustering for Big Data Mining. In *TrustCom/BigDataSE/ISPA*. IEEE.

[25] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to Play ANY Mental Game. In *STOC*. ACM.

[26] Q. Guo, X. Lu, Y. Gao, J. Zhang, B. Yan, D. Su, A. Song, X. Zhao, and G. Wang. 2017. Cluster Analysis: A New Approach for Identification of Underlying Risk Factors for Coronary Artery Disease in Essential Hypertensive Patients. In *Scientific Reports*.

[27] P. Hallgren, C. Orlandi, and A. Sabelfeld. 2017. PrivatePool: Privacy-Preserving Ridesharing. In *Computer Security Foundations (CSF)*. IEEE.

[28] K. Hamada, R. Kikuchi, D. Ikarashi, K. Chida, and K. Takahashi. 2012. Practically Efficient Multi-party Sorting Protocols from Comparison Sort Algorithms. In *International Conference on Information Security and Cryptology (ICISC)*. Springer.

[29] M. Huang, Q. Bao, Y. Zhang, and W. Feng. 2019. A Hybrid Algorithm for Forecasting Financial Time Series Data Based on DBSCAN and SVR. In *Information*.

[30] L. Hubert and P. Arabie. 1985. Comparing Partitions. In *Journal of Classification*. Springer.

[31] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. 2003. Extending Oblivious Transfers Efficiently. In *CRYPTO*. Springer.

[32] G. Jagannathan, K. Pillaipakkamnatt, R. Wright, and D. Umano. 2010. Communication-efficient Privacy-Preserving Clustering. In *Transactions on Data Privacy*. Springer.

[33] G. Jagannathan and R. N. Wright. 2005. Privacy-Preserving Distributed k-Means Clustering over Arbitrarily Partitioned Data. In *SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM.

[34] K. Järvinen, H. Leppäkoski, E. S. Lohan, P. Richter, T. Schneider, O. Tkachenko, and Z. Yang. 2019. PILOT: Practical Privacy-Preserving Indoor Localization using OuTsourcing. In *EuroS&P*. IEEE.

[35] A. Jäschke and F. Armknecht. 2018. Unsupervised Machine Learning on Encrypted Data. In *SAC*. Springer.

[36] S. Jha, L. Kruger, and P. McDaniel. 2005. Privacy Preserving Clustering. In *ESORICS*. Springer.

[37] D. Jiang, A. Xue, S. Ju, W. Chen, and H. Ma. 2008. Privacy-preserving DBSCAN on Horizontally Partitioned Data. In *International Symposium on IT in Medicine and Education*. IEEE.

[38] S. Kamara and M. Raykova. 2011. Secure Outsourced Computation in a Multi-Tenant Cloud. In *IBM Workshop on Cryptography and Security in Clouds*.

[39] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by Simulated Annealing. In *SCIENCE*.

[40] V. Kolesnikov and T. Schneider. 2008. Improved Garbled Circuit: Free XOR Gates and Applications. In *ICALP*. Springer.

[41] D. Kopanaki, N. Pelekis, A. Gkoulalas-Divanis, M. Vodas, and Y. Theodoridis. 2012. A Framework for Mobility Pattern Mining and Privacy- Aware Querying of Trajectory Data. In *Hellenic Data Management Symposium*.

[42] H.-P. Kriegel and M. Pfeifle. 2005. Density-Based Clustering of Uncertain Data. In *SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM.

[43] K. A. Kumar and C. P. Rangan. 2007. Privacy Preserving DBSCAN Algorithm for Clustering. In *Advanced Data Mining and Applications*. Springer.

[44] K. Kursawe, G. Danezis, and M. Kohlweiss. 2011. Privacy-friendly Aggregation for the Smart-Grid. In *PETS*. Springer.

[45] J.-G. Lee, J. Han, and K.-Y. Whang. 2007. Trajectory Clustering: a Partition-and-Group Framework. In *SIGMOD International Conference on Management of Data*. ACM.

[46] D. Liu, E. Bertino, and X. Yi. 2014. Privacy of Outsourced K-Means Clustering. In *ASIACCS*. ACM.

[47] J. Liu, L. Xiong, J. Luo, and J. Z. Huang. 2013. Privacy Preserving Distributed DBSCAN Clustering. In *Transactions on Data Privacy*.

[48] P. Mohassel and P. Rindal. 2018. ABY^3: A Mixed Protocol Framework for Machine Learning. In *CCS*. ACM.

[49] P. Mohassel, M. Rosulek, and N. Trieu. 2020. Practical Privacy-Preserving K-means Clustering. In *PoPETS*. Sciendo.

[50] D. Moulavi, P. A. Jaskowiak, R. J. G. B. Campello, A. Zimek, and J. Sander. 2014. Density-based clustering validation. In *International Conference on Data Mining*. SIAM.

[51] M. Naor and B. Pinkas. 1999. Oblivious Transfer and Polynomial Evaluation. In *STOC*. ACM.

[52] L. Ni, C. Li, X. Wang, H. Jiang, and J. Yu. 2018. DP-MCDBSCAN: Differential Privacy Preserving Multi-Core DBSCAN Clustering for Network User Data. In *IEEE Access*. IEEE.

[53] E. Pagnin, G. Gunnarsson, P. Talebi, C. Orlandi, and A. Sabelfeld. 2019. TOPPool: Time-aware Optimized Privacy-Preserving Ridesharing. In *PoPETS*. Sciendo.

[54] N. G. Pavlidis, V. P. Plagianakos, D. K. Tasoulis, and M. N. Vrahatis. 2006. Financial Forecasting through Unsupervised Clustering and Neural Networks. *Operational Research* (2006).

[55] N. Pelekis, A. Gkoulalas-Divanis, M. Vodas, A. Plemenos, D. Kopanaki, and Y. Theodoridis. 2012. Private-HERMES: A Benchmark Framework for Privacy-Preserving Mobility Data Querying and Mining Methods. In *Extending Database Technology*. ACM.

[56] G. Punj and D. W. Stewart. 1983. Cluster Analysis in Marketing Research: Review and Suggestions for Application. In *Journal of Marketing Research*.

[57] Y. Qi and M. J. Atallah. 2008. Efficient Privacy-preserving K-nearest Neighbor Search. In *International Conference on Distributed Computing Systems*. IEEE.

[58] M. S. Rahman, A. Basu, and S. Kiyomoto. 2017. Towards Outsourced Privacy-Preserving Multiparty DBSCAN. In *Pacific Rim International Symposium on Dependable Computing*. IEEE.

[59] D. Rathee, T. Schneider, and K. K. Shukla. 2019. Improved Multiplication Triple Generation over Rings via RLWE-Based AHE. In *CANS*. Springer.

[60] P. Rousseeuw. 1987. Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. In *Journal of Computational and Applied Mathematics*.

[61] S. Samet, A. Miri, and L. Orozco-Barbosa. 2007. Privacy Preserving K-means Clustering in Multi-Party Environment. In *SECRYPT*.

[62] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. 1998. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. In *SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM.

[63] A. Sangers, M. van Heesch, T. Attema, T. Veugen, M. Wiggerman, J. Veldsink, O. Bloemen, and D Worm. 2019. Secure Multiparty PageRank Algorithm for Collaborative Fraud Detection. In *FC*. Springer.

[64] U. Stemmer. 2020. Locally Private K-means Clustering. In *SIAM Symposium on Discrete Algorithms*. ACM.

[65] D. Su, J. Cao, N. Li, E. Bertino, and H. Jin. 2016. Differentially Private K-Means Clustering. In *Conference on Data and Application Security and Privacy*. ACM.

[66] D. Su, J. Cao, N. Li, E. Bertino, M. Lyu, and H. Jin. 2017. Differentially Private K-Means Clustering and a Hybrid Approach to Private Optimization. In *Transactions on Privacy and Security*. ACM.

[67] C. Troncoso, M. Payer, J.-P. Hubaux, M. Salathé, J. Larus, E. Bugnion, W. Lueks, T. Stadler, A. Pyrgelis, D. Antonioli, et al. 2020. Decentralized Privacy-Preserving Proximity Tracing. *IEEE Data Engineering Bulletin* (2020).

[68] A. Ultsch. 2005. Clustering wih som: U*c. In *Workshop on Self-Organizing Maps*.

[69] J. Vaidya and C. Clifton. 2003. Privacy-Preserving k-Means Clustering over Vertically Partitioned Data. In *SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM.

[70] N. X. Vinh, J. Epps, and J. Bailey. 2010. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *The Journal of Machine Learning Research* (2010).

[71] W. Wu, J. Liu, H. Wang, J. Hao, and M. Xian. 2020. Secure and Efficient Outsourced K-means Clustering using Fully Homomorphic Encryption with Ciphertext Packing Technique. In *Transactions on Knowledge and Data Engineering*. IEEE.

[72] W. M. Wu and H. K. Huang. 2015. A DP-DBScan Clustering Algorithm based on Differential Privacy Preserving. In *Computer Engineering and Science*.

[73] W. Xu, L.. Huang, Y. Luo, Y.. Yao, and W. W. Jing. 2007. Protocols for Privacy-Preserving DBSCAN Clustering. In *Int. Journal of Security and Its Applications*.

[74] A. C. Yao. 1986. How to Generate and Exchange Secrets. In *FOCS*. IEEE.

[75] S. Zahur, M. Rosulek, and D. Evans. 2015. Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gate. In *EUROCRYPT*. Springer.

A PLAINTEXT DBSCAN CLUSTERING

List. 4 shows the plaintext DBSCAN clustering as described in §3.1. For each unclassified data record *elem* (i.e., it was not assigned to a cluster yet) of the input data set, it is checked if *elem* has at least minPts neighbors (i.e., at least minPts elements in a radius of ϵ, cf. Line 9). If this is the case, a new cluster c containing *elem* and its neighbors is created in Lines 10 to 12. Otherwise, *elem* is considered as outlier and marked as noise (cf. Line 14). The function *expandCluster*(..) recursively iterates through all neighbors of *elem* and the neighbors of the neighbors to check if they are unclassified and core elements (cf. Line 21). If this is the case, they will also be added to cluster c.

Listing 4: Plaintext DBSCAN Clustering

```
1  Input: input, eps, minPts
2  Output: clusters
3
4  dbscan(vector data,eps,minPts):
5    vector clusters
6    for each elem in input:
7      if(elem.isUnclassified()): // no cluster assigned
8        neighbors = elem.getNeighbors()
9        if (|neighbors| >= minPts): // big enough?
10         c = new Cluster()
11         clusters.push(c)
12         expandCluster(elem,neighbors,eps,minPts,c)
13       else: // noise
14         elem[i].markAsNoise()
15   return clusters
16
17 expandCluster(p,queue,eps,minPts,c):
18   c.add(p)
19   while (queue.notEmpty()):
20     q = queue.pop()
21     if (q.isUnclassified() && |q.getNeighbors()|>minPts):
22       queue.append(q.getNeighbors())
23     if (q.isUnclassified() || q.isNoise()):
24       c.add(q)
```

B S2PC TECHNIQUES

Garbled Circuits. Yao's Garbled Circuits (GC) [74] allow to securely evaluate Boolean circuits between two parties. One party, called *garbler*, encrypts the gates of the circuit using random keys for each wire and sends it (now called *garbled circuit*) together with the keys associated to his inputs to the other party, called *evaluator*. The evaluator receives the keys associated to his inputs via Oblivious Transfer (OT) [7, 31, 51] from the garbler. After decrypting the garbled circuit, the parties jointly decode the output keys. ABY includes state-of-the-art optimizations like Free-XOR [40], fixed-key AES [11], and Half-Gates [75]. XOR gates are for free and AND gates cost 2κ bits of communication in an input-independent setup phase, where $\kappa = 128$ is the symmetric security parameter [17].

Arithmetic Sharing. Arithmetic sharing uses additive shares of l-bit integers in the ring $\mathbb{Z}_{2^l}$. To share the secret value x, the data owner (who can be one of the parties) chooses a random value $r \in_R \mathbb{Z}_{2^l}$ and sets $\langle x \rangle_i^A = x - r$ and $\langle x \rangle_{i-1}^A = r$. Party P_i receives share $\langle x \rangle_i^A$ and party P_{i-1} gets $\langle x \rangle_{i-1}^A$. When provided with both shares, any party can reconstruct the secret by addition of the shares. While secure addition can be executed locally, secure multiplication with arithmetic shares requires interaction and multiplication triples [10] that can be efficiently precomputed with OTs [17] or homomorphic encryption [59].

Conversions. Converting an Arithmetic share to Yao sharing (A2Y) costs $6l\kappa$ communication bits and $12l$ AES operations [17]. Yao to Arithmetic (Y2A) sharing conversion costs $l\kappa + (l2 + l)/2$ and $6l$ AES operations. Both conversions require two messages. l is the bit-length of the Arithmetic share.

C SECURITY DISCUSSION

In this section, we sketch why ppDBSCAN (§4.3) and ppTRACLUS (§4.4) are privacy-preserving, i.e., a semi-honest adversary learns nothing beyond what can be inferred from the output of the protocols. Generally, the security of ppDBSCAN and ppTRACLUS follows directly from the provable security of the used S2PC techniques (cf. §B): Yao's Garbled circuits (GC) [74] and Arithmetic sharing [25].

At the beginning of the protocol, the data owners secret-share their input data records among them (2PC) or among two non-colluding parties (Outsourcing [38]). Hence, those have only access to indistinguishable random secret-shares that do not leak anything about the input data. Parameters can similarly be input as random secret-shares that do not leak any information. In the first step, the distance calculation (cf. §4.3.2 for ppDBSCAN and §4.4 for ppTRACLUS), no secret-shares are opened and only conversions between Arithmetic sharing and Yao's GC are run which are provably secure [17]. Similar reasoning applies for the second phase, the clustering (cf. §4.3), if *maxIterations* is set to the data set size n. To enhance efficiency, *maxIterations* can be fixed or it can be checked in a privacy-preserving manner using secure computation if the vector *elem.neighbors* has changed. This, however, leaks one bit of information implying some information about the data distribution (e.g., how elongated clusters can be).

To summarize, ppDBSCAN and ppTRACLUS can be instantiated fully privacy-preserving such that all information remain secret-shared during the execution of the entire protocol.

D BENCHMARK RESULTS

In this section, we provide more details and additional results of our benchmarks of ppDBSCAN and ppTRACLUS.

D.1 Additional Experiments for Clustering Quality of ppTRACLUS

Tab. 5 presents the results of TRACLUS and ppTRACLUS on the Deer data set. It contains the number of clusters created from the Deer data set which are equal with TRACLUS and ppTRACLUS. When only one cluster is found, SC and DBCV cannot be computed. Nevertheless, having only one cluster does not necessarily mean that the quality of the clustering algorithm is low. It simply says that the algorithm found only one group of similar elements.

Tab. 6 contains the results of Experiment 2 of TRACLUS, ppTRACLUS, and ppTRACLUS' on the Travel data set.

Table 5: Clustering quality assessment for TRACLUS and ppTRACLUS on the Deer data set. For all scores larger values are better (best marked in bold).

Data Set	Score	Experiment 1		Experiment 2	
		TRACLUS	ppTRACLUS	TRACLUS	ppTRACLUS
Deer	**(ϵ', minLns)**	$(400, 3)$	$(1 \times 10^6, 3)$	$(282, 3)$	$(550 \times 10^3, 3)$
	# of Clusters	1	1	2	2
	Noise	1	480	20	1333
	SC	N/A	N/A	0.089	**0.36**
	SC_{noise}	N/A	N/A	0.089	**0.34**
	DBCV	N/A	N/A	0.47	**0.79**

Table 6: Results of Experiment 2 of the clustering quality assessment for TRACLUS, ppTRACLUS, and ppTRACLUS' on the Travel data set. The best results are marked in bold.

	Experiment 2		
	TRACLUS	ppTRACLUS	ppTRACLUS'
(ϵ', minLns)	$(47 \times 10^3, 3)$	$(13.5 \times 10^9, 3)$	$(13.5 \times 10^9, 3)$
# of Clusters	1	1	2
Noise	0	359	361
SC	N/A	N/A	**0.82**
SC_{noise}	N/A	N/A	**0.81**
DBCV	N/A	N/A	**0.98**

D.2 Runtime and Communication Costs

Tab. 7 contains the runtimes (in seconds) and Tab. 8 contains the communication costs (in MB) of the SED, the approx. distance of TRACLUS, and clustering averaged over 10 experiments. The SED distance is applied on 2-dimensional data, the approx. TRACLUS distance on 4-dimensional line segments. *maxIterations* is set to 4.

Table 7: Runtimes for ppDBSCAN/ppTRACLUS.

Data Set Size n	Runtimes (s)		
	Squared Euclidean Distance	Appr. TRACLUS-Distance	Density-based Clustering
100	2.36	4.36	10.9
200	8.04	14.82	58.43
300	17.35	28.94	171.12
400	29.03	45.39	391.69
500	44.17	64.98	750.284
600	61.04	86.91	1317.70
700	79.76	113.07	2043.01
800	102.99	144.09	2970.52
900	125.82	177.48	4187.54
1000	157.07	208.92	5682.425

Table 8: Communication Costs for ppDBSCAN/ppTRACLUS.

Data Set Size n	Communication (MB)		
	Squared Euclidean Distance	Appr. TRACLUS-Distance	Density-based Clustering
100	144	328	460
200	575	1317	3566
300	1294	2968	11912
400	2300	5280	28090
500	3594	8253	54694
600	5176	11889	94315
700	7044	16185	149548
800	9201	21143	222983
900	11645	26762	317214
1000	14376	33043	434835