Noname manuscript No.
(will be inserted by the editor)

Enabling Search Services on Outsourced Private Spatial Data

Man Lung Yiu · Gabriel Ghinita · Christian S. Jensen · Panos Kalnis

Abstract Cloud computing services enable organizations and individuals to outsource the management of their data to a service provider in order to save on hardware investments and reduce maintenance costs. Only authorized users are allowed to access the data. Nobody else, including the service provider, should be able to view the data. For instance, a real-estate company that owns a large database of properties wants to allow its paying customers to query for houses according to location. On the other hand, the untrusted service provider should not be able to learn the property locations and, e.g., selling the information to a competitor.

To tackle the problem, we propose to transform the location datasets before uploading them to the service provider. The paper develops a spatial transformation that re-distributes the locations in space, and it also proposes a cryptographic-based transformation. The data owner selects the transformation key and shares it with authorized users. Without the key, it is infeasible to reconstruct the original data points from the transformed points. The proposed transformations present distinct trade-offs between query efficiency and data confidentiality. In addition, we describe attack models for studying the security properties of the transformations. Empirical studies demonstrate that the proposed methods are efficient and applicable in practice.

Man Lung Yiu
Department of Computing, Hong Kong Polytechnic University
E-mail: csmlyiu@comp.polyu.edu.hk

Gabriel Ghinita
Department of Computer Science, Purdue University
E-mail: gghinita@cs.purdue.edu

Christian S. Jensen
Department of Computer Science, Aalborg University
E-mail: csj@cs.aau.dk

Panos Kalnis
Division of Mathematical and Computer Sciences and Engineering, KAUST University
E-mail: panos.kalnis@kaust.edu.sa

Keywords: Data Outsourcing, Spatial Query Processing

1 Introduction

Cloud computing services enable individuals and organizations to outsource the management of their data with ease and at low cost, even if they lack IT expertise. Cloud computing enables scalability with respect to storage and computational resources as the number of service requests grows, without the need for costly investments in hardware and maintenance.

Motivating Applications.
Consider the example of a real-estate company that owns a large database with descriptions of properties and their locations. The company (i.e., the *private data owner*) wishes to allow *authorized users* (e.g., paying customers) to query for properties situated within a certain geographical region. To save on hardware investments and maintenance costs, the data owner *outsources* the management of its dataset to a *service provider (SP)* that specializes in data storage and query processing. However, the SP may not be fully trusted, and could sell the data to a competitor. Furthermore, even if the SP is trusted, a malicious attacker can compromise the SP and gain unauthorized access to the data. To prevent such attacks, the data owner first encrypts the dataset according to a secret transformation and then uploads the encrypted data to the SP. Only authorized users who know the transformation are able to learn the property locations.

The illustrated scenario is relevant for other applications as well: for instance, a research institute that deploys a large sensor network is able to outsource collected data (which include the geo-spatial tag of the originating sensor), without disclosing the locations of the sensors to the SP.

Outdoor advertising companies determine the pricing for billboards placed in geo-referenced spots such as at bus stops, on buildings or along roads. Significant human effort is spent

on assessing the pricing of billboards so the constructed dataset is valuable to its owner. In the outsourcing context, the owner allows its dataset to be queried only by business partners who are contractually obligated not to share the data with others.

Marketing companies employ databases for managing various demographic information (e.g., income, age range) of geo-referenced households. Such databases can be rather large. These companies wish to outsource the management of their data while allowing their paying users to access the data. For instance, a paying user could be a financial company that may want to access the data in order to run targeted advertising and marketing campaigns (e.g., by direct mail).

Problem Scenario.
The paper focuses on the outsourcing of spatial datasets. Our objective is to enforce the user authorization specified by the data owner, even when the service provider cannot be trusted. The paper presents techniques that protect *location data* from attackers, while allowing authorized users to issue spatial queries that are executed efficiently by the SP. Any non-spatial content (e.g., text, images, etc.) is assumed to be encrypted using conventional encryption, and it can be decrypted by authorized users. Given a set P of data points, the data owner maps P to another point set P' using a transformation with a *secret key*. The data owner uploads P' to the SP and sends the key to authorized users through a secure channel. Since the SP does not know the key, it cannot derive P. At query time, an authorized user $\mathcal{U}$ maps a query q to another query q' by using the key and then submits q' to the SP. Next, the SP executes q' against P' and returns the result $R' \subseteq P'$ to $\mathcal{U}$, who uses the key to decode R' and obtain the actual result $R \subseteq P$.

Challenges.
A brute-force solution is to apply conventional encryption (e.g., AES [2]) to the whole dataset P and then store the encrypted file at the server. This solution is secure because the server is unable to learn any information from the encrypted file. At query time, the whole encrypted file is downloaded to the client, decrypted, and searched for the requested results. This solution is inefficient for typical queries that only require access to a small fraction of the data.

We require that the transformation method must satisfy two essential criteria: (*i*) it should be infeasible to reconstruct the precise points of P from P' without the key, and (*ii*) it should support efficient and accurate computation of range queries. Consider for instance a simple transformation that maps the x and y coordinate of each point in P to $x' = a_1x+a_2$ and $y' = b_1y+b_2$, respectively (see Figure 1). Suppose that the key is $\langle a_1, a_2, b_1, b_2\rangle$. The above transformation satisfies the second condition, since $\mathcal{U}$ can easily map a query rectangle q to a rectangle q', and the SP can employ any spatial index to answer q' efficiently. However,

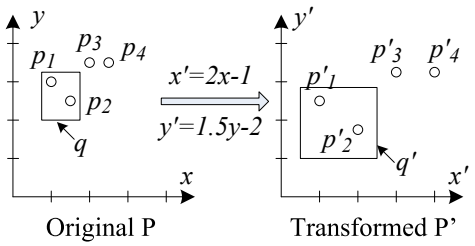


Fig. 1 Example of an Insecure Transformation

the first condition is not satisfied. An attacker who learns the original and transformed versions of 4 non-collinear points can compute the key by solving a system of 4 linear equations; therefore, the database is compromised. We call this a *tailored attack*, since it relies on knowledge about the actual transformation technique used. For certain transformations, the tailored attack is computationally hard. In this case, the attacker may resort to performing a *general attack*, which is a transformation-independent method of estimating the approximate locations of original points. The paper contributes definitions of these two attack models, which are then used to evaluate the security of the proposed transformations.

Contributions.
The paper substantially extends preliminary work [37]. Table 1 summarizes the trade-offs of our proposed techniques in terms of data security and query efficiency. Each entry with 'N/A' indicates that the attack that the column concerns is computationally infeasible to perform on the transformation technique that the row concerns.

First, we propose *spatial transformations* that protect data by altering the spatial distribution. Two preliminary building blocks in this category are Hierarchical Space Division (HSD) and Error-Based Transformation (ERB). HSD utilizes a spatial partitioning technique for redistributing the transformed data; it is strong against the general attack but weak against the tailored attack. ERB exploits a secure hash function for injecting noise into the data; it is immune to the tailored attack, but it incurs high query cost. Thus, we design an advanced method called Enhanced HSD (HSD*) that integrates the desirable features of HSD and ERB.

Second, we develop a Cryptographic Transformation (CRT) technique that completely prevents the disclosure of location data. CRT is the most secure against all attacks; however, it requires multiple communication rounds during query evaluation.

Recently, there has been substantial interest in database outsourcing. Most related work [3, 9, 17, 18] assumes that a tamper-proof device (or trusted software) exists in front of the SP. This device holds the decryption keys and performs the encryption and decryption operations during query processing. Since the device resides at the SP, there is no communication cost, so transformations can be complex. In our

Table 1 Trade-offs Among the Proposed Transformation Techniques

Method	Tailored attack	General attack	Transferred data cost	Round trips
HSD (preliminary)	Feasible	Ineffective	Low	1
ERB (preliminary)	N/A	Effective	High, rises as ϵ	1
HSD*	N/A	Ineffective	Low	1
CRT	N/A	N/A	Moderate	Tree height

setting, it is not feasible for every data owner to install her own secure device at the SP. If query processing requires on-the-fly transformations, these must be done by the user; therefore, the techniques must achieve low communication cost.

Finally, the term "database outsourcing" has been used to denote the process of authenticating the correctness and completeness of query results, without hiding user-generated content from the SP. This issue is orthogonal to our work; any related method [36] can be used on top of our transformations. Our work is also orthogonal to spatial anonymity methods [12,13,28], which protect the location of the querying user, but do not hide the content being queried from the SP.

In summary, our contributions are:

- We present a framework for protecting the confidentiality of spatial data that is outsourced to a service provider, while supporting the efficient processing of spatial queries (i.e., range and k nearest neighbor queries) on the data.
- We develop tailored and general attack models for assessing the security of the transformations.
- We propose two transformation techniques (HSD* and CRT) that represent different trade-offs between data security and query efficiency.
- We present an extensive experimental evaluation of our techniques using real datasets.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the general problem setting. Section 4 introduces the attack models and presents the spatial transformations. Section 5 presents the cryptographic transformation. An empirical evaluation is presented in Section 6. Finally, Section 7 concludes and discusses future research directions.

2 Related Work

Outsourced Databases.
The outsourcing of database services to an SP was introduced by Hacigümüs et al. [18]. The confidentiality of outsourced data was subsequently addressed by Hacigümüs et al. [17]. In their solution, the SP stores the encrypted tuples together with auxiliary *bucketing* information to facilitate indexing. This bucketing technique returns a superset of the actual query results, which calls for potentially expensive filtering.

Agrawal et al. [3] point out that bucketing is vulnerable to *estimation exposure*, i.e., an attacker can infer the approximate values of the encrypted data. Motivated by these limitations, they propose an order-preserving encryption scheme (OPES) [3] for one-dimensional numeric values (e.g., salaries) that transforms the input data distribution (e.g., Zipfian) into a user-specified target distribution (e.g., Gaussian). While this scheme supports efficient processing of range queries at the SP, it assumes that an attacker cannot stage a known plaintext attack. In contrast, our techniques support the general case where an attacker knows a sample of the data points as well as their encrypted values (i.e., plaintext attack).

In a proposal by Damiani et al. [9], the data owner builds a B^+-tree on one-dimensional data and then encrypts each node using conventional cryptographic techniques (e.g., AES). Since such encryption does not preserve order, the decryption key is required in order to process queries. A trusted front-end at the SP is assumed. In practice, a tamper-resistant device can be used for this purpose. Our CRT technique uses a similar approach in conjunction with an R^*-tree (for two-dimensional data). However, in our case, the SP is likely to serve a large number of data owners, each with a distinct encryption key. Furthermore, the SP may offer the service for free, rendering it financially infeasible to employ one tamper-resistant device per data owner. For this reason, the SP and the user must employ a multi-stage protocol during query processing; therefore, we must minimize the communication cost.

Wong et al. [35] study the computation of k nearest neighbor queries on encrypted tuples stored at an untrusted server. The encryption key is a $(d+1) \times (d+1)$ invertible matrix, where d is the data dimensionality. Their encryption scheme enables the server to decide, among two encrypted data points p'_1 and p'_2, the one closer to the encrypted query point q', with respect to the original space. They propose two optimizations to strengthen the security: (i) splitting each data point (or query point) into two random parts, and (ii) inserting artificial random dimensions into the dataset. To achieve high security, Wong et al. [35] suggest to use 80 artificial dimensions. However, there is no effective index for (encrypted) points in the resulting high dimensional space, due to the dimensionality curse [33]. Thus, at query time, the server is forced to scan the entire dataset. In contrast, our proposed solutions simply re-distribute the locations of points in the two-dimensional space. Any spatial index (e.g., the R-tree) can be built on the transformed dataset for facilitating efficient query processing.

A related issue in data outsourcing is the *authentication* of query results, which must be *sound* (the SP does not alter the data), and *complete* (the SP returns all valid results). The *Merkle Hash Tree* (MH-tree) [27] is a main memory binary tree for indexing 1D values that has been applied for the authentication of range queries [10]. Disk-based indices for authenticating the results of multi-dimensional range queries have been proposed recently [8, 36]. Authentication is orthogonal to our problem (i.e., confidentiality), and existing authentication schemes can be used on top of our transformations.

Privacy-Preserving Data Publication.
Confidentiality has also been addressed in the context of privacy-preserving publication of sensitive datasets. In this setting, the data owner (e.g., a hospital) wishes to release data records (e.g., for research purposes) without violating the privacy of individuals concerned. Even though directly identifying data (e.g., names) are not disclosed, the records still contain quasi-identifier (QID) attributes, such as Age or $ZipCode$, which can be used by an attacker to re-identify records. The k-anonymity principle [30] addresses this threat by requiring each tuple to be indistinguishable from at least $k-1$ other tuples, with respect to the QID. To achieve k-anonymity, it is common to generalize QID values [14, 23]. More recent data privacy paradigms include ℓ-diversity [26], which prevents association between QID values and sensitive attributes (e.g., medical condition), and t-closeness [24], which attempts to reproduce the overall distribution of the sensitive attributes in each class (whose tuples have the same generalized QID). In all these methods, the generalization process alters the dataset in a way that prevents exact query answers, rendering the data useful only for statistical purposes.

Papadimitriou et al. [29] apply perturbation techniques on time-series data before they are published. Correlated noise is added into the time series such that (i) the utility of the perturbed data is bounded by a specified threshold, and (ii) the error between the adversary's reconstructed data and the original data is maximized. They consider two attack models: filtering and true value leakage. These attacks are analogous to the noise removal attack and the general attack that we will discuss in Section 4.1, respectively. The above technique is inapplicable to our problem because there is no key for accurately reconstructing the original data from the perturbed data.

The above-mentioned privacy-preserving publication methods belong to the category of *input perturbation* techniques. More recently, Dwork et al. [11] introduced the concept of *differential privacy*, which is an *output perturbation* technique. In this setting, the data are never published. Instead, a statistical database management system answers *aggregate* queries (e.g., count, sum) on top of the data, perturbing the query results to preserve privacy. Given any two datasets P and P' of equal cardinality and that differ in exactly one tuple, a statistical database is said to satisfy ϵ-indistinguishability if for any transcript (i.e., sequence of queries) $\mathcal{Q}$ and the associated result $\mathcal{R}$, it holds that $Pr(\mathcal{Q}(P) = \mathcal{R}) \leq \exp(\epsilon) \cdot Pr(\mathcal{Q}(P') = \mathcal{R})$. In other words, an attacker is not able to learn whether the result $\mathcal{R}$ was obtained by answering $\mathcal{Q}$ from P or from P'. The property of ϵ-indistinguishability is achieved by adding random noise to the result of each aggregate query. The differential privacy model is not suitable to our problem for two reasons: First, data outsourcing requires disclosing the entire (transformed) dataset to the SP, whereas in the differential privacy model, the data owner only discloses results to a restricted set of queries. Second, differential privacy addresses only approximate aggregate queries, as opposed to our model where the actual data points (and exact locations) must be returned to the authorized users.

In privacy-preserving data mining, a mining task (e.g., classification, clustering) is outsourced to the service provider for extracting "hidden patterns" from the *perturbed data*, without the original data being revealed. For instance, various data perturbation techniques [4, 25] can be utilized for introducing noise into the data. These techniques do not guarantee exact answers over the perturbed data and therefore are inapplicable to our problem.

Location Privacy.
Privacy preservation of location data has been studied in the context of spatial queries. Mobile users issue spatial queries such as "find the French restaurant nearest to my location," which are answered by a public service (e.g., Google Maps). Users do not want to reveal their exact location to the service. Therefore, a user location is first generalized according to the spatial k-anonymity principle, often by employing a trusted location anonymizer [12, 16, 20, 28]. The anonymizer maintains the locations of all subscribed users, and upon receiving a query from a user, it constructs an Anonymizing Spatial Region (ASR) that encloses the locations of that user and $k-1$ other users. The service processes the query based on the ASR, instead of the exact user location. Observe that spatial k-anonymity hides the query user's location from the service, but does not protect the data being queried; therefore, it is orthogonal to our problem.

Other approaches to enabling private nearest-neighbor queries [13, 22, 38] adopt privacy definitions that differ from spatial k-anonymity. One solution [22] employs the Hilbert curve mapping [7] for transforming the set of data points into 1D values. In a preprocessing stage, a trusted entity transforms each point p_i into its Hilbert value $H(p_i)$ and then uploads the values to the service. The parameters of the transformation (curve orientation, scale, etc.) act as the encryption key and are kept secret from the service. In addition, users possess tamper-resistant devices containing the key. Upon issuing a query q, the user computes $H(q)$ and re-

quests the closest data value (in terms of 1D Hilbert values) from the service. By applying the inverse mapping H^{-1} to the value received, the user obtains the answer. This solution is approximate in nature and does not provide any guarantee on the result accuracy. In addition, the evaluation of range queries remains to be addressed. A possible solution is to decompose a region query W into multiple 1D Hilbert intervals and issue these separate interval queries to the service. The disadvantages are: (*i*) the number of decomposed intervals can be large, leading to high processing and communication costs, and (*ii*) the decomposition of intervals provides hints for an attacker to reverse engineer the transformation.

3 Problem Setting

This section introduces formally the problem of confidential spatial data outsourcing. We focus on two-dimensional point datasets, which are the most common type of spatial data. Table 2 summarizes the notation used. In the sequel, we focus on the most popular query—the range query.

Table 2 Summary of Notations

Symbol	Description
P	Original point set
P'	Transformed point set
p_i, p'_i	Point belonging to P, P'
$p.x, p.y$	X, Y-coordinate of point p
$dist(p_i, p_j)$	Euclidean distance between points p_i and p_j
Υ	Number of parameters in transformation key
S	Subset of points in P known by the attacker
S'	Set of transformed points from S
$\mathcal{V}(p, S)$	Feature vector of p with respect to set S

Definition 1 Range Query. Given a rectangular query range $W = [x_l, x_h] \times [y_l, y_h]$ and a set P of points, the range query retrieves each $p \in P$ that intersects with W.

We assume the architecture of Figure 2. In a pre-processing phase, the data owner chooses a secret *transformation key*, and then converts the original point set P into the transformed point set P' (step 1.1). Next (step 1.2), the data owner builds a multi-dimensional index $\mathcal{IND}(P')$ over P'; this step is optional. The transformed dataset is sent to the SP. The data owner trusts all her *authorized users* and sends them the transformation key (step 1.3) over a secure communication channel (e.g., SSL). Note that the key size is much smaller than the size of P. Furthermore, since the data owner trusts the users, expensive tamper-resistant devices are not necessary.

To issue a query q, the user $\mathcal{U}$ encodes q to q' using the key and sends q' to the SP (step 2.1). The SP evaluates the query over P' and returns the encoded results to $\mathcal{U}$ (step 2.2). At this point, if the optional authentication information exists, $\mathcal{U}$ can verify the correctness and completeness of the result. Finally, $\mathcal{U}$ decodes the results using the key, thus obtaining the original points.

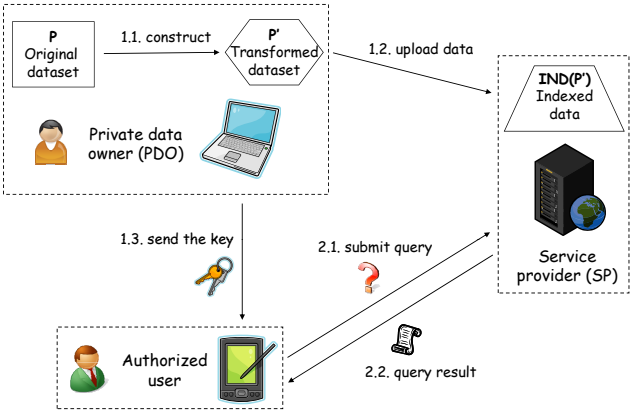


Fig. 2 Our Framework for Private Spatial Data Outsourcing

We envision that the service provider will offer services by utilizing existing cloud computing infrastructure. Depending on the setting, the revenue of the service provider may come via different combinations of advertisement revenues, storage fees (paid by the data owners), and query fees (paid by the query users).

Our objective is to develop transformation techniques that satisfy the following criteria:

1. It must be hard for an attacker (including the SP) to recover precisely the original dataset P from the transformed dataset P'.
2. The computational overhead of the transformation method should be low, so that the users can efficiently encode queries and decode query results.
3. The encoded result set returned by the SP must cover all the actual results. The number of false positives in the encoded result set should be as small as possible.

4 Spatial Transformations

We first formulate the notion of *spatial transformation*. For the sake of discussion, we assume that the spatial domain is the unit square $[0, 1]^2$, for both the original point set P and the transformed point set P'. Given a point $p = (x, y)$ in P, we compute its transformed point $p' = (x', y')$ in P', as follows:

$$x' = F_X(x, y) \qquad y' = F_Y(x, y), \tag{1}$$

where $F_X(\cdot)$ and $F_Y(\cdot)$ are the transformation functions for the x and y coordinate, respectively. These functions are associated with a hidden *transformation key*, which will be elaborated upon for each specific transformation technique. The *key length* Υ indicates the number of parameters used in the key. In addition, we define the inverse functions $F_X^{-1}(\cdot)$ and $F_Y^{-1}(\cdot)$, which use the same transformation key to decode the original point $p = (x, y)$ from the transformed

point $p' = (x', y')$:

$$x = F_X^{-1}(x', y') \qquad y = F_Y^{-1}(x', y') \tag{2}$$

As an example, consider a simple geometric transformation (GEO), the transformation key of which consists of four parameters: the rotation angle $\theta \in [0, 2\pi)$, the scaling factor λ, and translations X_t, Y_t (along the X-axis and Y-axis, respectively). The original point $p = (x, y)$ in P is converted into the transformed point $p' = (x', y')$ in P' as follows:

$$\begin{aligned} x' &= F_X(x, y) = (x \cos\theta - y \sin\theta) \cdot \lambda + X_t \\ y' &= F_Y(x, y) = (x \sin\theta + y \cos\theta) \cdot \lambda + Y_t \end{aligned} \tag{3}$$

It is easy to define the inverse functions $F_X^{-1}(\cdot)$ and $F_Y^{-1}(\cdot)$, which also depend on θ, λ, X_t, and Y_t. In terms of query processing, given a range query $W = [x_l, x_h] \times [y_l, y_h]$ in the original space, we apply the functions $F_X(\cdot)$ and $F_Y(\cdot)$ to transform the four corners of W, i.e., (x_l, y_l), (x_h, y_l), (x_h, y_h), (x_l, y_h), and obtain (x'_l, y'_l), (x'_h, y'_l), (x'_h, y'_h), (x'_l, y'_h). By connecting the transformed points (using straight lines) in the anti-clockwise direction, we obtain a new query W' that can be efficiently processed at the SP with any spatial index. In terms of security, however, GEO can easily be compromised (to be seen shortly).

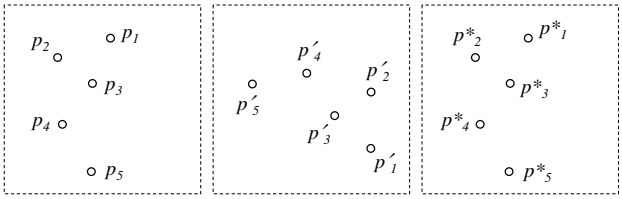
Section 4.1 presents several attack models for spatial transformations. Section 4.2 proposes the HSD transformation, which applies a spatial partitioning technique for redistributing the transformed data. Section 4.3 discusses the ERB transformation, which introduces bounded errors into the data that are reversible with the help of the key. These two transformations are preliminary methods that are used as building blocks. Section 4.4 uses them for obtaining a hybrid solution called HSD*, which avoids the disadvantages of HSD and ERB. For each technique, we study three aspects: (i) the transformation of data points, (ii) the transformation of range queries, and (iii) the analysis of attacks. Finally, in Section 4.5, we discuss how to evaluate k nearest neighbor queries in the context of our proposed transformation methods.

4.1 Attack Models

This section introduces the concept of estimation distortion and then studies a wide range of attack models for spatial transformation methods.

Let P be the original dataset and P' be the transformed dataset. We assume that the attacker knows the dataset P' but not P. The attacker may obtain some background knowledge, e.g., a subset of mappings between P and P' (to be discussed shortly). Then, the attacker applies some attack method (to be studied soon) to estimate the original locations of the objects as the estimated dataset P^*.

Estimation Distortion.
We define the *estimation distortion* between the original dataset P and the estimated dataset P^* as:



(a) original set P (b) transformed set P' (c) estimated set P^*

Fig. 3 Sets of data points

$$\mathcal{DT}(P, P^*) = \texttt{AVG}_{p.id=p^*.id, p\in P, p^*\in P^*}\ dist(p, p^*), \tag{4}$$

where $dist(p, p^*)$ denotes the Euclidean distance between p and p^*.

A low $\mathcal{DT}(P, P^*)$ value suggests that the transformation method is vulnerable to attacks. Figure 3a shows the points in the original dataset P. After applying the GEO transformation on P, we obtain the transformed dataset P' as shown in Figure 3b. As we will see later, the attacker can break GEO easily and estimate the locations of the original points as the dataset P^* shown in Figure 3c. It is worth noticing that $\mathcal{DT}(P, P^*)$ is very small, even though the distances between the points in P and P' are large.

Thus, the estimation distortion $\mathcal{DT}(P, P^*)$ serves as an intuitive measure of the security of a transformation method.

Attacks without Background Knowledge.
Kargupta et al. [21] propose a *noise removal attack* method that aims at reconstructing approximately the original dataset from the transformed dataset P', without needing prior background knowledge of any actual point in the original dataset. This attack is effective in recovering original data from perturbed data [21].

For the purpose of data privacy analysis, the data owner can apply the above attack to P' to obtain a reconstructed dataset P^* and then measure the distortion $\mathcal{DT}(P, P^*)$ between P and P^*. A transformation technique is said to be vulnerable to an attack if $\mathcal{DT}(P, P^*)$ is small. We will study experimentally the effect of the above noise removal attack on our transformation techniques.

Attacks with Background Knowledge.
We introduce a new attack model that is analogous to the *known-plaintext attack* in the cryptography literature[1]. In practice, an attacker may acquire prior background knowledge of the dataset. For example, in the scenario outlined in Section 1, the attacker may know that a neighbor registered a property for sale (i.e., one data point is known) and, in the worst case, predicts correctly its transformed point. Even though the attacker has some prior knowledge of the dataset, it is worth preventing him from learning any meaningful in-

[1] This attack was not considered in the work of Agrawal et al. [3].

formation (e.g., distribution, dense regions, outliers) about the remaining data points.

Recall that P and P' are the original and transformed datasets, respectively. Assume the attacker knows the following information:

- A set $S \subset P$ of m points, $S = \{s_1, s_2, ..., s_m\}$.
- The set $S' \subset P'$ of transformed points, $S' = \{s'_1, s'_2, ..., s'_m\}$, where s'_i is the transformed point of s_i.

In the above example, the attacker passively knows the subsets S and S'; the attacker cannot actively choose S (and S') as he wishes. Note that although the attacker knows each transformed location in $P' - S'$, he does not know the location of any point in $P - S$. The attacker attempts to infer the original location of each point in $P' - S'$. Our goal is to protect the original data points.

Based on the above attack model, we present two instances: a tailored attack and a general attack.

Tailored Attack.
If the attacker knows the precise transformation technique used by the data owner to encode points, he can devise a specific attack for the transformation, which is defined as follows.

Definition 2 Tailored Attack. Given the known subsets S and S', the tailored attack is a technique specific to the transformation F (and F^{-1}) such that it *computes* the *exact* transformation key value.

To recover the original dataset, the attacker may determine the values of the key parameters by solving a system of equations. As an example, we apply the tailored attack to the GEO transformation discussed earlier. Assume the attacker knows a set $S \subset P$ of 4 non-collinear points and the corresponding set S' of points in P'. Since Equation 3 utilizes only four parameters, the attacker can solve the system of four equations corresponding to the known points to obtain the exact parameter values. Thus, GEO is insecure.

General Attack.
In scenarios where the tailored attack is computationally hard to perform (see Sections 4.3 and 4.4), the attacker may attempt an attack independent of the transformation. We define this attack as follows:

Definition 3 General Attack. Given the known subsets S and S', the general attack is a transformation-independent method that *estimates* an *approximate* original location of some transformed point in $P' - S'$.

We propose a (heuristic-based) general attack method that allows the attacker to estimate a reasonable approximation of the original data, by exploiting his limited knowledge of known points. Formally, for a point $p' \in P' - S'$, we define its *feature vector* over S' as:

$$\mathcal{V}(p', S') = \langle dist(p', s'_1), dist(p', s'_2), ..., dist(p', s'_m) \rangle,$$

where $dist(\cdot)$ denotes the Euclidean distance. Given a location c in the original domain space, its feature vector over S is:

$$\mathcal{V}(c, S) = \langle dist(c, s_1), dist(c, s_2), ..., dist(c, s_m) \rangle.$$

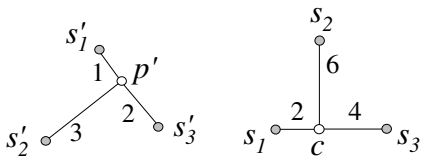
An attacker can identify the candidate point c as the original location of the transformed point p' if the feature vectors $\mathcal{V}(p', S')$ and $\mathcal{V}(c, S)$ exhibit coherent patterns. To allow direct comparison, we normalize $\mathcal{V}(p', S')$ and $\mathcal{V}(c, S)$ by their magnitudes $|\mathcal{V}(p', S')|$ and $|\mathcal{V}(c, S)|$, respectively. Then, the *dissimilarity* between c and p' is defined as:

$$\Phi(c, p') = L_1 \left(\frac{\mathcal{V}(p', S')}{|\mathcal{V}(p', S')|}, \frac{\mathcal{V}(c, S)}{|\mathcal{V}(c, S)|} \right),$$

where L_1 is the Manhattan distance[2]. Based on this concept, the attacker *estimates* the original location of p' as p^*, which is defined as the location c having the smallest $\Phi(c, p')$ value:

$$p^* = \text{argmin}_c \, \Phi(c, p'). \quad (5)$$

Consider an example of this general attack method. Figure 4a illustrates a transformed space with $S' = \{s'_1, s'_2, s'_3\}$. The attacker attempts to estimate the original location of a point $p' \in P' - S'$. The lines connecting the points are labeled with the corresponding Euclidean distances. The attacker calculates $\mathcal{V}(p', S') = \langle 1, 3, 2 \rangle$. Figure 4b depicts the original space with $S = \{s_1, s_2, s_3\}$ and a location c. The attacker derives $\mathcal{V}(c, S) = \langle 2, 6, 4 \rangle$, and computes the value of $\Phi(c, p')$, which equals 0. Since $\Phi(c, p')$ reaches the minimum value, the attacker estimates $p^* = c$ to be the original location of p'.



(a) transformed space (b) original space

Fig. 4 General Attack Example

Observe that Equation 5 is not in closed form, as the derivation of p^* takes into account an infinite number of candidate locations c in the original space. In our experiments, we assume that the attacker applies a randomized numerical method (e.g., the Monte Carlo method) to obtain an accurate approximation of p^* within reasonable time.

By applying the general attack on the transformed dataset P', the attacker obtains a reconstructed dataset P^*:

$$P^* = \{ \text{ argmin}_c \, \Phi(c, p') \mid p' \in P' \}.$$

[2] We chose the L_1 norm because it is robust for multi-dimensional feature vectors. We have also experimented with the L_2 and L_∞ norms, finding that L_1 yields a higher success probability for the attacker (i.e., lower estimation error).

The attacker's estimation error is defined as the distortion $\mathcal{DT}(P, P^*)$ between P^* and the original dataset P.

Note that the attacker cannot compute $\mathcal{DT}(P, P^*)$, since he does not know P. On the other hand, the data owner is able to conduct a "what-if" analysis for the value $\mathcal{DT}(P, P^*)$, based on specific instances of S and S'. For example, "What is the average estimation error of the attack, if the attacker already knows the exact locations of 10 points of P?"

Intuitively, the attacker's estimation error decreases as the set S increases. Furthermore, the distribution of the points in S (in the original space) can also influence the estimation error. If the points in S are close to each other, there is a considerable amount of redundant information in the feature vector $\mathcal{V}(c, S)$, leading to a poor estimation by the attacker. In Section 6, we study these issues empirically.

We now demonstrate that the GEO transformation mentioned earlier is completely vulnerable to the general attack. Even though the distances between the original dataset P and the transformed dataset P' are large, it does not mean that GEO is secure against the general attack. Let p' be a transformed point in $P' - S'$ and let p be the original point. The following lemma shows that $\Phi(p, p')$ is always 0, for any distance-preserving transformation function (e.g., GEO). Thus, the attacker would estimate the location of p as the original point of the transformed point p'. In other words, the attacker can completely reconstruct the original dataset P^* because the estimation distortion $\mathcal{DT}(P, P^*)$ is zero.

Lemma 1 *Let Ξ be a distance preserving transformation function. Let $S \subset P$ be a set of actual points and its corresponding transformed point set be $S' \subset P'$. Let p' be a transformed point in $P' - S'$ and let p be the original point of p'. It holds that $\Phi(p, p') = 0$.*

Proof Since the function Ξ preserves the exact distances among the points in the transformed space, we obtain: $dist(p, s_i) = dist(p', s'_i)$ for each $s_i \in S$. Thus, we derive $\Phi(p, p') = 0$.

Discussion.
The focus of our work is to protect outsourced datasets against location-based attacks, i.e., attacks that rely on spatial information. Such an approach is common for many related works on privacy of location data, e.g., private spatial queries (e.g., [12, 16, 22, 38]). Note that other types of attacks may exist, such as *timing-based* or *frequency-based* attacks. For instance, consider the example from Section 1 of the real-estate company that outsources its datasets of properties. Following a public announcement of, e.g., the construction of a new metro station, the customer interest in a particular region can increase sharply. Therefore, a malicious attacker may infer that the large number of points queried immediately after the announcement are nearby the metro construction site. Furthermore, the frequency of data point accesses can also disclose information about their location. For instance, points in the central city areas, or famous touristic landmarks are more likely to be queried by users. Both of the above-mentioned threats are possible because the malicious attacker is able to observe the access pattern of the data points. Techniques targeted at protecting the privacy of access patterns have been reported previously in the literature. For instance, oblivious RAM (random-access memory) solutions have been proposed [15, 34]. Such techniques rely on re-encrypting and re-shuffling the dataset after a certain number of accesses. A similar re-shuffling can be applied to our transformations. However, such extensions are beyond the scope of this paper.

4.2 Preliminary Method: HSD Transformation

The *Hierarchical Space Division* (HSD) is a spatial transformation that hides the data by altering the distribution of the points in the transformed space. This way, the attacker cannot get any insight into the original data through inspecting the data distribution in the transformed space. HSD achieves this goal by means of a kD-tree partitioning of the data points.

Data Point Transformation.
The transformation key of HSD contains $2(2^E - 1)$ parameters, where E is an even integer specifying the granularity. The original point set from the data owner is denoted by P. A target point set T is used to capture the data distribution in the transformed space. In its simplest form, T can be a uniform dataset. Alternatively, T can be set to a publicly known data distribution (e.g., known maps of other cities) in order to mislead the attacker.

Algorithm 1 shows the HSD pseudo-code for extracting key parameters from P and T. Initially, the transformed space domain $(x'_l, x'_h] \times (y'_l, y'_h]$ is set to $(0, 1] \times (0, 1]$. In

Algorithm 1 HSD Key Parameter Generation

execute HSD_Key($E, P, T, (0, 1] \times (0, 1]$);

Algorithm HSD_Key(Level E, Point set P, Target set T, Transformed space $(x'_l, x'_h] \times (y'_l, y'_h]$)
1: create a root node Z;
2: **if** $E \mod 2 = 0$ **then**
3: $Z.v$:=the median X value of points in P;
4: $Z.v'$:=the median X value of points in T;
5: **if** $E > 0$ **then** ▷ the recursive case
6: P_L:={ $p \in P \mid p.x \leq Z.v$ };
7: P_H:={ $p \in P \mid p.x > Z.v$ };
8: T_L:={ $t \in T \mid t.x \leq Z.v'$ };
9: T_H:={ $t \in T \mid t.x > Z.v'$ };
10: Z_L:=HSD_Key($E - 1, P_L, T_L, (x'_l, Z.v'] \times (y'_l, y'_h]$);
11: Z_H:=HSD_Key($E - 1, P_H, T_H, (Z.v', x'_h] \times (y'_l, y'_h]$);
12: set Z_L and Z_H as the left/right child nodes of Z;
13: **else**
14: apply Lines 3–12, but for Y values of points in P and T;
15: **return** Z;

Line 1, a root node Z is created. If E is an even number, only the X coordinate values are processed (Lines 3–12); otherwise, the Y coordinate values are considered (Line 14). In Lines 3–4, we record for node Z the following values: (i) $Z.v'$, the median X value of the points in T, and (ii) $Z.v$, the median X value of the points in P. When E is non-zero (Line 5), the set P is divided into two sets, such that P_H contains all tuples of P with X values above $Z.v$, whereas P_L has the remaining tuples of P. Similarly, the set T is divided into two sets T_L and T_H. The algorithm is first performed on P_L and T_L recursively to obtain the left child node (i.e., Z_L) of the current node Z. Then, the algorithm is applied on P_H and T_H recursively to obtain the right child node (i.e., Z_H) of Z. Finally, the current node Z is returned to the caller.

The output of Algorithm 1 is an E-level tree with $2^E - 1$ nodes, where each node Z stores the splitting X-values (or Y-values) $Z.v$ and $Z.v'$ for the original and the transformed space, respectively. Clearly, the tree offers a partitioning of both spaces. Figure 5a shows the original point set P and Figure 5b depicts the target point set T. The corresponding transformation tree with $E = 2$ levels is shown in Figure 5c.

To transform an original point $p = (x, y)$ into $p' = (x', y')$, it suffices to follow one path of the tree (i.e., the path leading to p). During the traversal, we maintain the rectangle $A = [Ax_l, Ax_h] \times [Ay_l, Ay_h]$ that contains p in the original space. We also derive the corresponding rectangle $A' = [A'x_l, A'x_h] \times [A'y_l, A'y_h]$ that encloses p' in the transformed space. The transformed point $p' = (x', y')$ is determined as:

$$x' = F_X(x) = A'x_l + (A'x_h - A'x_l) \cdot \frac{x - Ax_l}{Ax_h - Ax_l}$$

$$y' = F_Y(y) = A'y_l + (A'y_h - A'y_l) \cdot \frac{y - Ay_l}{Ay_h - Ay_l} \quad (6)$$

Figure 5d depicts the location of points in P' after performing the above transformation. For instance, the leaf rectangle $A = (0, 0.4] \times (0, 0.7]$ containing the original point p_3 corresponds to rectangle $A' = (0, 0.5] \times (0, 0.4]$ containing the transformed point p'_3. Observe that the distribution of points in P' becomes more similar to T and different from P.

To decode a transformed point $p' = (x', y')$, we also apply the transformation procedure described above. The only difference is that we compute the rectangle A' that contains p' in the transformed space and then search the tree in order to derive the corresponding rectangle A in the original space. The original point p is determined as:

$$x = F_X^{-1}(x') = Ax_l + (Ax_h - Ax_l) \cdot \frac{x' - A'x_l}{A'x_h - A'x_l}$$

$$y = F_Y^{-1}(y') = Ay_l + (Ay_h - Ay_l) \cdot \frac{y' - A'y_l}{A'y_h - A'y_l} \quad (7)$$

Both the encoding and the decoding of a point take $O(E)$ time, since E is the height of the transformation tree.

Range Query Transformation.
The transformation of a range query W is similar to that of a point, except that all branches of the tree intersecting W are followed. In this case, we may obtain multiple rectangles A_i in the original space along with their corresponding rectangles A'_i in the transformed space. Figure 6a depicts three example queries, W_1, W_2, and W_3, in the original space. The transformed queries are depicted in Figure 6b. Leaf rectangles in the original and transformed spaces are labeled A_i and A'_i, respectively. For instance, W_1 intersects only A_1, so it is transformed to region W'_1 inside rectangle A'_1. Next, W_2 intersects both A_3 and A_4, so it corresponds to two regions in A'_3 and A'_4. To avoid leaking any partitioning information to the attacker, these two regions are not used directly. Instead, their minimum bounding rectangle (MBR) is taken as the transformed range query W'_2. Query W_3 is also transformed into two regions in A'_2 and A'_4. Again, we take their MBR as the transformed range query W'_3. In this case, W'_3 contains extra space; therefore, some false positives may be retrieved from the service provider. These false positives are discarded by the user after transforming the result into the original space.

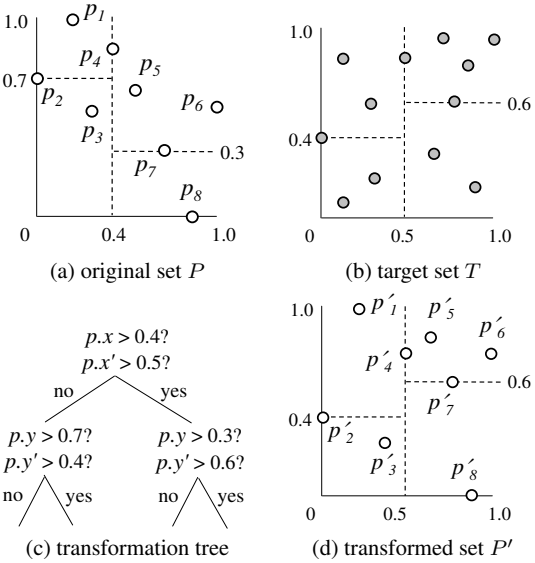


Fig. 5 HSD Transformation Example, $E = 2$

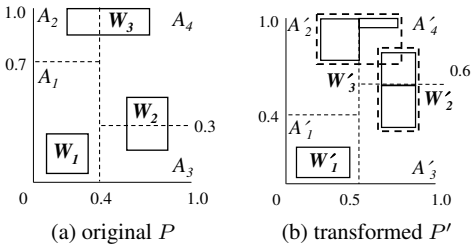


Fig. 6 HSD Query Example, $E = 2$

Note that the false positives only affect the query evaluation and communication costs, but do not affect the correctness of the final result. Our experiments show that the num-

ber of false positives tends to be negligible when the query rectangle is small compared to the area of a space partition.

Analysis of Tailored Attack.
Recall that the attacker needs to know a subset $S \subseteq P$ of points before launching the attack. The attacker substitutes the coordinates of each known point $p \in S$ (and the transformed $p' \in S'$) in Equation 6. The partitioning rectangle of p' (i.e., the values of $A'x_l$, $A'x_h$, $A'y_l$, $A'y_h$) can be derived immediately from p' provided that the attacker also knows the value of E. The values of the unknowns Ax_l, Ax_h, Ay_l, Ay_h can be found if the attacker knows another point of S in the same partitioning rectangle as p.

In the HSD transformation key, there are $2^E - 1$ independent key parameters obtained from the distribution of P. To break the HSD transformation, the attacker needs to acquire a known subset $S \subseteq P$ with at least $2^E - 1$ points such that the subset S contains at least a point for each partition. Such a worst-case scenario is unlikely to occur. In case most of the points of S are located in the same partition, the attacker can only break the local transformation in that partition, not in the other partitions.

A nice feature of HSD is that all partitions contain the same number of points, i.e., the maximum number of points in any partition is minimized. This minimizes the success probability of the tailored attack.

4.3 Preliminary Method: ERB Transformation

In the literature, perturbation techniques [4, 25] have been applied to inject noise into datasets that are outsourced to service providers for performing data mining tasks. However, this process is irreversible—individual original points cannot be reconstructed perfectly from the outsourced data points. We present an error-injection technique that is reversible by the data owner (and query users), but is computationally infeasible to compromise for an attacker.

A *secure hash function* converts an arbitrary-length plaintext message into a fixed-length digest message. It is computationally infeasible for the attacker to (*i*) recover the original message from the digest message, or (*ii*) find another plaintext message for generating the same digest message. The SHA-512 function, approved as a NIST standard [1], computes a 512-bit digest message from its input message. This function is readily deployable in existing systems. Our *Error-Based Transformation* (ERB) is built on top of such a secure hash function. The core idea is that by dividing the digest message with its domain size, we obtain a real number in $[0, 1)$ that can be viewed as a pseudo-random number generated deterministically from the original message.

Data Point Transformation.
In ERB, the data owner specifies a transformation key consisting of three parameters: an error threshold $\epsilon \in [0, 1)$ and two 512-bit key values $\mathcal{K}_X$ and $\mathcal{K}_Y$ (whose length is the security strength of the hash function). We assume that each point has a unique identifier id. Given an original data point $p = \langle id, x, y \rangle$, its transformed point $p' = \langle id, x', y' \rangle$ is computed as follows:

$$x' = (1 - \epsilon) \cdot x + \epsilon \cdot \texttt{SHA}(\mathcal{K}_X \circ id)$$
$$y' = (1 - \epsilon) \cdot y + \epsilon \cdot \texttt{SHA}(\mathcal{K}_Y \circ id), \quad (8)$$

where SHA returns a real number in $[0, 1]$ and $\circ$ denotes concatenation.

Table 3 shows an example of the ERB transformation with error threshold $\epsilon = 0.1$. Each value generated by SHA is underlined. Although the locations of the points with IDs id_1 and id_2 coincide in the original space, their transformed locations differ (due to their different IDs). The transformed locations for the points with IDs id_1 and id_3 are near to each other although their original locations are not close. Indeed, ERB resembles a potential many-to-many mapping, which is fundamentally different from the one-to-one mapping of HSD.

Table 3 ERB Transformation Example, $\epsilon = 0.1$

Original Point	**Transformed Coordinates**	
$\langle id, x, y \rangle$	x'	y'
$\langle id_1, 0.4, 0.7 \rangle$	0.9·0.4 + 0.1·<u>0.8</u> = 0.44	0.9·0.7 + 0.1·<u>0.1</u> = 0.64
$\langle id_2, 0.4, 0.7 \rangle$	0.9·0.4 + 0.1·<u>0.3</u> = 0.39	0.9·0.7 + 0.1·<u>0.8</u> = 0.71
$\langle id_3, 0.5, 0.6 \rangle$	0.9·0.5 + 0.1·<u>0.0</u> = 0.45	0.9·0.6 + 0.1·<u>0.9</u> = 0.63

Knowing the threshold ϵ and the key values $\mathcal{K}_X$ and $\mathcal{K}_Y$, it is trivial to use the ID of a point p' and its transformed location (x', y') to reconstruct its original location (x, y):

$$x = (\ x' - \epsilon \cdot \texttt{SHA}(\mathcal{K}_X \circ id)\ )/(1 - \epsilon)$$
$$y = (\ y' - \epsilon \cdot \texttt{SHA}(\mathcal{K}_Y \circ id)\ )/(1 - \epsilon) \quad (9)$$

Range Query Transformation.
To guarantee that range queries are evaluated correctly, it suffices to convert an original query $W = [x_l, x_h] \times [y_l, y_h]$ into a transformed query $W' = [x'_l, x'_h] \times [y'_l, y'_h]$ as follows:

$$x'_l = (1 - \epsilon) \cdot x_l + \epsilon \cdot 0$$
$$x'_h = (1 - \epsilon) \cdot x_h + \epsilon \cdot 1 \quad (10)$$

This is so because any SHA value is bounded between 0 and 1. The values of y'_l and y'_h are derived similarly.

Upon receiving the result of the transformed query W' (from the server), the client decodes each transformed result point p' back into p and then checks whether p is an actual result. Observe that there may exist false positives (but not false negatives) among the points returned by the server. The parameter ϵ enables trade-offs between the data distortion and the number of false positives. For large ϵ, the transformed data undergo substantial distortion, but the number of false positives increases.

Analysis of Attacks.
Recall that the attacker knows a subset $S \subset P$ and the corresponding subset $S' \subset P'$. In case S is large, it is possible

for the attacker to derive an approximate value of the error threshold ϵ (e.g., by employing linear least-squares fitting). For the tailored attack, the attacker substitutes the above known points and the ϵ value into Equation 8, obtaining the SHA values. These are then exploited to recover their original messages, in order to discover the keys $\mathcal{K}_X$ and $\mathcal{K}_Y$. However, it is impractical to perform this attack against SHA-512 with existing computational resources [1].

Regarding the general attack, the attacker's estimation error is roughly at the value ϵ. The reason is that each original point coordinate is mapped to a value in an ϵ-length interval in the transformed space.

Communication Cost Analysis.
The parameter ϵ affects the (encoded) result size. To simplify our analysis, we assume that the original dataset contains n data points that are uniformly distributed in the unit square $[0,1]^2$. We assume that the range query q is a square with side length r. Let the original query be $W = [x_l, x_h] \times [y_l, y_h]$ and the transformed query be $W' = [x'_l, x'_h] \times [y'_l, y'_h]$.

According to Theodoridis and Sellis [32], the actual number of (original) data points that fall into the query range W is derived as:

$$\mathcal{RS}_{OPT} = n \cdot r^2$$

Based on the assumption of the square query, we have:

$$x_h - x_l = y_h - y_l = r$$

According to the transformation of a range query (see Equation 10), we obtain the side length of the transformed query:

$$x'_h - x'_l = (1-\epsilon)r + \epsilon$$

Since the original dataset follows a uniform distribution, the transformed dataset also follows a uniform distribution. Thus, the number of transformed data points that fall into the transformed query range W' is derived as:

$$\mathcal{RS}_{ERB} = n \cdot (r + \epsilon(1-r))^2 \tag{11}$$

From the above equation, we observe that the communication overhead of ERB (over the optimal cost) becomes low when ϵ is small or r is large.

4.4 Enhanced Method: HSD* Transformation

The HSD transformation technique partitions the space and then applies a distinct linear transformation function for each partition. However, an attacker who knows two points in the same partition can infer the precise transformation function used in that partition. Although the ERB transformation is resistant against the above attack, it incurs high query costs for typical values of ϵ.

We develop a novel transformation called *Enhanced HSD* (HSD*) that exploits the strengths of HSD and ERB: It applies the HSD transformation for the global space and then performs the ERB transformation at the level of partitions.

Transformation of Data Points and Range Queries.
In HSD*, the transformation key consists of $2(2^E - 1)$ partitioning values, as in HSD, and an error threshold ϵ together with two 512-bit keys $\mathcal{K}_X$ and $\mathcal{K}_Y$, as in ERB.

First, we apply Algorithm 1 to decompose the space into 2^E disjoint partitions. To transform an original point $p = \langle id, x, y \rangle$ into $p' = \langle id, x', y' \rangle$, it suffices to find the rectangle $A = [Ax_l, Ax_h] \times [Ay_l, Ay_h]$ that contains (x, y) in the original space, and the corresponding rectangle $A' = [A'x_l, A'x_h] \times [A'y_l, A'y_h]$ that encloses (x', y') in the transformed space. Value x' is then computed as follows:

$$x' = A'x_l + (A'x_h - A'x_l) \cdot \left((1-\epsilon) \cdot \frac{x - Ax_l}{Ax_h - Ax_l} + \epsilon \cdot \mathtt{SHA}(\mathcal{K}_X \circ id) \right) \tag{12}$$

Value y' is computed similarly.

For the transformation of a range query W, we follow the procedure presented in Section 4.2. The only difference is that we now apply Equation 12 in each local partition to convert each subquery (in a partition) into a subquery in the transformed space.

Analysis of Attacks.
We now examine the tailored attack on the HSD* method. As in HSD, the attacker formulates a system of equations by substituting the coordinates of known points. However, even if the attacker knows two points in the same partition, the SHA values in the transformed points prevent the leakage of the transformation key in that partition. HSD* inherits its security strength from ERB and is thus computationally hard to break, as explained in Section 4.3.

4.5 Supporting k Nearest Neighbor Search

We proceed to develop a method for processing k nearest neighbor (kNN) queries on transformed data. It is designed to guarantee the retrieval of exact kNN results. Also, the proposed method is generic and thus applicable to all the transformations (HSD, ERB, HSD*) described earlier.

Transformed Incremental kNN Search.
Algorithm 2 shows the pseudo-code of a client-side method for performing kNN search on transformed data at the server. The querying user specifies the query point q and the number k of required results. In addition to that, the algorithm also needs to know the transformation method $\mathcal{TRN}$ and the key value $\mathcal{K}$ used for the transformation.

First, it employs a result heap $\mathcal{RH}$ for maintaining the k points closest to q seen so far. The variable γ denotes the top distance (i.e., the largest one) stored in $\mathcal{RH}$. The query

point q is converted to a query point q' in the transformed space. The client then issues the incremental nearest neighbor search (INN) query [19] to the server, in order to retrieve (transformed) points in ascending order of their distances from q'. It is worth noticing that we replace the distance metric used in INN by the L_∞ norm in order to simplify geometric comparisons that we will encounter shortly. In Line 7, the point p' is retrieved from the server as the next closest point to q' (in the transformed space). The variable τ represents the largest L_∞ distance from q' of points seen. We then define a square region SQ' with q' as its center and 2τ as its side length. The property of INN search guarantees that any (transformed) point in SQ' has been retrieved.

Next, the client decodes the transformed point p' back to its original point p. In case q is closer to p than some existing point in the result heap $\mathcal{RH}$, we update $\mathcal{RH}$ and the kNN distance γ by using p. We then formulate another square region W with q as its center and 2γ as its side length. Observe that the region W is guaranteed to cover the actual kNN of q, regardless of whether the actual kNN results have been retrieved or not. In Line 14, we apply the transformation to convert W into a set of rectangles B. The loop in Lines 6–15 terminates if the searched region SQ' covers each rectangle of B, as the actual kNN of q must then have been retrieved. After that, the client terminates the incremental NN query at the server and reports the points of $\mathcal{RH}$ as the result to the query user.

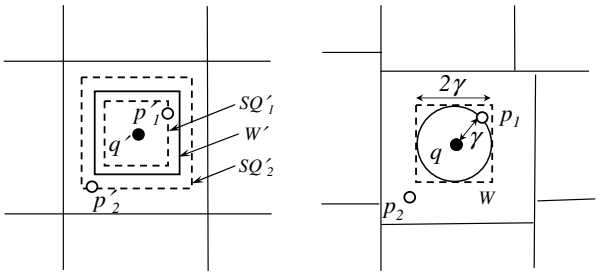
Algorithm 2 Transformed Incremental kNN Search Method at Client-Side

```
   Algorithm Transformed_Key(Query point q, Value k, Transformation TRN, Key K)
 1: RH:=new max-heap of pairs <p, dist(q,p)>;
 2: insert k dummy pairs <NULL, ∞> into RH;
 3: γ:=the top distance of RH;        ▷ kNN distance of q (seen so far)
 4: point q':=apply the transformation on the query point q;
 5: issue an incremental NN query at q' to the server (using the L∞
      norm as the distance metric);
 6: repeat
 7:     transformed point p':=retrieve the next closest point to q' from
          the server;
 8:     τ:=L∞(q', p');              ▷ furthest L∞ distance seen from q'
 9:     SQ':=a square region with center as q' and side length as 2τ;
10:     point p:=apply the inverse transformation to decode p';
11:     if dist(q,p) < γ then
12:         update RH and γ by using the point p;
13:     W:=a square region with center as q and side length as 2γ;
14:     Set of rectangles B:=apply transformation on W;
15: until SQ' covers W'_i, for each W'_i ∈ B
16: terminate the incremental NN query at the server;
17: return RH as the result;
```

Example.
We illustrate the running steps of the above algorithm using an example. Assume that the user wishes to perform a kNN search at the query location q, with $k = 1$. Figures 7a,b show the locations of points in the transformed space and the original space, respectively.

First, the client computes the transformed query point q' and issues the incremental NN query at q' to the server (using the L_∞ norm as the distance metric), in Figure 7a. The server reports the point p'_1. Based on the property of INN search, we conclude that any (transformed) point in the square region SQ'_1 has been retrieved. The client then computes the corresponding original point p_1 from p'_1 and updates the NN to be p_1 (see Figure 7b). The current NN distance γ is $dist(q, p_1)$. Then the square region W is defined by its center q and side length 2γ. It is guaranteed that W contains the actual NN of q.

The client then converts W to the rectangle W' in the transformed space, as shown in Figure 7a. Since the region SQ'_1 does not contain W', the search continues. Next, the server retrieves the point p'_2 as the next closest point to q'. The region SQ'_2 (defined by p'_2) now contains W', meaning that it is not possible to find other points in W' in the future. The client algorithm terminates and returns p_1 as the actual NN of q.



(a) transformed space (b) original space

Fig. 7 Example of Transformed Incremental kNN Search, at $k = 1$

It is worth noticing that the server only knows the transformed query point q', the retrieved points p'_1 and p'_2 (and their associated square regions SQ'_1 and SQ'_2). The server does not know the original query point q and the rectangles W and W' (computed at the client side).

5 Cryptographic Transformation

This section presents our cryptographic transformation technique and discusses how to minimize the query processing cost incurred by this technique.

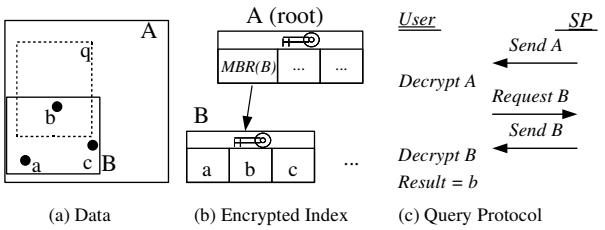
5.1 CRT Transformation

The *Cryptographic Transformation* (CRT) employs conventional cryptographic techniques (e.g., AES [2]) to achieve data confidentiality. CRT provides provable confidentiality

guarantees, inherited from the encryption technique. The advantage of CRT is that spatial information is completely obscured in the transformed data, thwarting any type of location-based attack (such as the general attack). However, query processing at the SP is rendered difficult.

Range Search.
CRT employs R*-trees, and it is a substantial extension over the encrypted B$^+$-trees of Damiani et al. [9]. The objective of CRT is to let the user learn the exact query result, while minimizing the communication cost between the user and the SP. In contrast to the setting assumed by Damiani et al. [9], our architecture does not require the existence of a tamper-resistant device at the SP. This creates the additional challenge of answering queries through a distributed, multiple-round protocol between the query user and the SP.

The functionality of CRT is exemplified in Figure 8. Data points (e.g., a, b, c) are stored in an encrypted index (only the relevant part is shown). To find the result for query



(a) Data (b) Encrypted Index (c) Query Protocol

Fig. 8 Query Processing in CRT

q, the encrypted root (node A) is sent to user $\mathcal{U}$, who decrypts A and determines that the MBR of node B intersects q. Then $\mathcal{U}$ retrieves node B from the SP and computes the query result (i.e., point b). Every index node that intersects q must be sent to the user. The protocol operates in this level-by-level manner, and the number of communication rounds equals the tree height.

Assuming an average node fan-out f, the user needs to retrieve f pairs of encrypted coordinates for each node; therefore, the communication cost is proportional to f. However, a large f translates into a lower index height and a smaller number of accessed internal nodes. Therefore, an optimal choice of f exists that minimizes the communication cost. Observe that CRT is likely to incur higher communication cost than HSD, especially for queries with small extent. The reason is that CRT will always return at least one *entire* leaf node. If, for instance, the result set contains a single point, CRT returns the entire node, incurring a cost proportional to f.

kNN Search.
As before, we apply the state-of-the-art incremental nearest neighbor algorithm [19] for performing the kNN search on top of CRT.

Let $mindist(q, e)$ denote the minimum distance between a query point q and an R-tree entry e [19]. At the client side, a min-heap H is employed for organizing entries of the tree to be visited in the ascending order of their (minimum) distances from q. The client first requests the encrypted root node from the server, and then it decrypts the node and inserts all entries of the node into H.

In each iteration, the client deheaps the top entry e_{cur} of H, and checks whether $mindist(q, e_{cur})$ is greater than the best kNN distance found so far. If so, the client terminates the search, as it is guaranteed to have found the actual kNN objects. Otherwise, the client requests the child node of e_{cur} from the server and then decrypts the node. In case the retrieved node is a leaf node, the entries in the node are used to update the kNN objects found so far. If the retrieved node is a non-leaf node, its entries are inserted into the min-heap H. The above procedure is repeated until the termination condition (discussed earlier) is satisfied.

As a remark, the number of communication rounds of this search method is equal to the number of node requests sent from the client to the server.

5.2 Analysis of Communication Cost

To determine the optimal fan-out f for the R*-tree, we employ the analysis of Böhm [5] to determine the expected number of nodes accessed by a range query, assuming a uniform data distribution. Given a generic index, i.e., a partitioning of n data points into a set of nodes represented as hyper-rectangles, each with capacity f, and a (square) range query q with side r, the estimated number of nodes intersected by q is $\left(r\sqrt{\frac{n}{f}} + 1 - \frac{1}{f}\right)^2$. By applying this at each level of the R*-tree, the total number of nodes accessed by q becomes:

$$Cost(f, r, n) = \sum_{i=1}^{\lceil \log_f n \rceil} \left(r\sqrt{\frac{n}{f^i}} + 1 - \frac{1}{f^i}\right)^2 \qquad (13)$$

Therefore, the communication cost is $f \cdot Cost(f, r, n)$. We can determine the f value that minimizes the cost using numerical methods. The end-to-end user time is computed as:

$$RTT \cdot \lceil \log_f n \rceil + f \cdot Cost(f, r, n) \cdot (\vartheta_T + \vartheta_D),$$

where RTT is the round-trip network time and ϑ_T and ϑ_D are the average data transfer time and decryption time per entry.

6 Experimental Study

Table 4 summarizes the techniques and parameter values used in the study. Unless stated otherwise, their default values, shown in bold, are used. In addition to our proposed

solutions (HSD*, CRT), we cover two preliminary solutions (HSD, ERB) and also introduce two benchmark methods in our comparison. Thus, BULK is a secure solution that stores the whole encrypted dataset at the server; when a query is issued, the client retrieves the entire dataset in a brute-force manner. In contrast, OPT is an insecure solution that stores the original dataset at the server and has optimal communication cost (i.e., no false positives).

Table 4 Experimental Parameter Settings

Parameter	Values
Method	BULK, OPT, HSD, ERB, HSD*, CRT
Dataset	OL, TG, **NE**, NA, SF
Target set (for HSD/HSD*)	OL, TG, **UI**, NA, SF
Key size Υ (for HSD/HSD*)	64, 256, **1024**, 4096
Error ϵ (for ERB/HSD*)	0.05, 0.10, **0.15**, 0.20, 0.25
Node capacity (for CRT)	5, 10, 20, **50**, 100, 200
Query window extent	0.005, 0.01, **0.02**, 0.05, 0.1
k	1, 2, **5**, 10, 20
$\|S\|$	20, 50, **100**, 200, 500
Covering radius of S	0.05, 0.10, **0.25**, 0.50, 1.00

We evaluate the techniques using four real spatial datasets: Oldenburg (OL: $6,105$ points), San Joaquin County (TG: $18,263$ points), San Francisco (SF: $174,956$ points), North America (NA: $175,813$ points), and North East USA (NE: $123,593$ points). The datasets OL, TG, and SF were obtained from Brinkhoff et al. [6], the dataset NA was available at the Digital Chart of the World[3], and the dataset NE was obtained from the R-tree portal[4]. In particular, the points in the NE dataset correspond to real postal addresses in New York, Philadelphia, and Boston, so the dataset matches well the real-estate company outsourcing application covered in the introduction.

The domain of each dataset is normalized to the unit square $[0,1] \times [0,1]$. The target set (for HSD/HSD*) is used to specify the distribution of points in the transformed space; it is fixed to the uniform independent distribution (UI) by default. The key size Υ (for HSD/HSD*) denotes the number of values in the spatial transformation key. ERB/HSD* requires an error threshold value ϵ, whereas CRT uses a node capacity parameter.

Regarding the query distribution, we generate each range query as a randomly distributed square region with a default side length of 2% of the domain space extent. The query point of each kNN query is randomly generated from the domain space, and the default value of k is 5. We assess the query performance in terms of *communication cost*, which is the dominant cost component. The reported cost is taken as the average over 100 experimental instances. We represent each identifier (ID) by 4 bytes, and each point coordinate in double precision IEEE 754 format (8 bytes). Hence, a data point takes 20 bytes (i.e., ID and 2 coordinates). In the CRT method, each non-leaf MBR entry takes 36 bytes (i.e., ID and 4 coordinates). The cost of CRT includes the messages sent from the SP to the user (i.e., encrypted blocks), as well as those sent from the user to the SP (i.e., requests for individual data blocks). For some experiments, we report the *end-to-end-user time*, which captures not only the communication time, but also the round trip delay and the decryption time at the client side.

We consider the robustness of our methods against various attacks, in which the attacker knows a subset $S \subset P$ of the original points and its transformed set $S' \subset P'$. S is generated by picking a random subset of P, based on two parameters: the cardinality $|S|$ and the covering radius r of S (i.e., the radius of the minimum enclosing circle of S). The general attack is applicable to HSD, ERB, and HSD*, which employ spatial transformations of the data. Here, we measure the security in terms of the *attacker's estimation error*, as mentioned in Section 4.1. The tailored attack requires the attacker to derive the exact locations of original points, so it is applicable only to HSD. In this case, we measure the security by the *leakage percentage*, i.e., the fraction of key parameters derivable by the attacker (the lower the better). Recall from Sections 4.3 and 4.4, that it is computationally infeasible to launch the tailored attack against ERB and HSD*. CRT is used in conjunction with the 256-bit key AES [2] encryption scheme, and therefore it inherits the provable security guarantees of AES. CRT is vulnerable to neither tailored nor general attacks based on spatial information.

6.1 Visualization and Attacks

We visualize the proposed transformations using the real spatial point set NA. Figure 9a shows the original point set. Note that the Western and Eastern parts have comparatively high densities. Figure 9b.1 depicts the data transformed by HSD, with $\Upsilon = 1024$, i.e., $E = 10$. The transformed data distribution is completely different from the original one. Figure 9c.1 illustrates the data transformed by ERB, at $\epsilon = 0.15$. The distortion achieved is weaker than that of HSD. Nevertheless, it is able to hide particular characteristics from the original dataset, e.g., the 'holes' in the West, the exact densities of dense regions, and outliers (in the Northwest, South, and Southeast). We then apply the general attack (with the default parameter values) on these transformed datasets obtained from HSD and ERB. The corresponding estimated datasets are shown in Figures 9b.2 and c.2, respectively. The attacker learns vaguely that few of the points are distributed at the boundary of the map. However, the attacker cannot precisely reconstruct the exact locations of all original data points.

Next, we study whether a dataset transformed by ERB (i.e., a perturbation method) can be exploited by the noise

[3] www.maproom.psu.edu/dcw

[4] www.rtreeportal.org

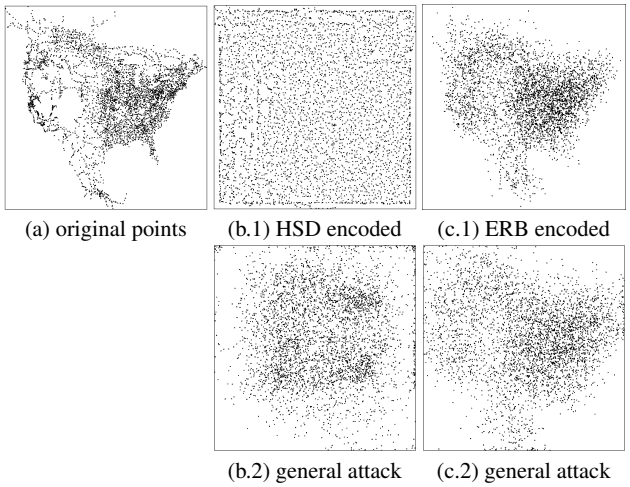


Fig. 9 Visualization of North America Points (NA)

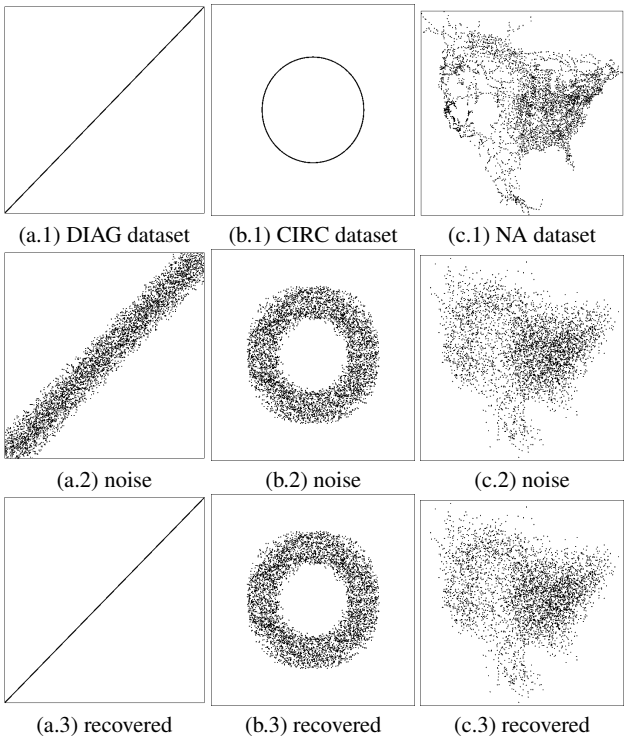


Fig. 10 Visualization of Perturbed Points and Recovered Points

removal attack [21] for reconstructing the original dataset. In addition to the real dataset NA, we introduce two synthetic datasets for the sake of comparison: (i) the DIAG dataset (in Figure 10a.1) contains 5,000 points following a diagonal pattern, and (ii) the CIRC dataset (in Figure 10b.1) contains 5,000 points following a circular pattern.

Figures 10a.1, a.2, a.3 show the original DIAG dataset, the transformed DIAG dataset (by ERB), and the reconstructed dataset from the transformed data, respectively. The noise removal attack successfully reconstructs the original dataset in this case. Figures 10b.1, b.2, b.3 illustrate the original CIRC dataset, the transformed CIRC dataset (by ERB), and the reconstructed dataset, respectively. The noise removal attack fails to reconstruct the original data from the transformed data, even though the original data follows only a simple distribution. Figures 10c.1, c.2, c.3 depict the original NA dataset, the transformed NA dataset (by ERB), and the reconstructed dataset, respectively. The ERB transformation successfully protects the NA dataset against the noise removal attack.

The robustness of ERB against the attack described by Kargupta et al. [21] is explained as follows: previous perturbation techniques (which are easily compromised by noise filtering) do not use secret transformation keys; therefore, data processing and noise filtering have equal capabilities. To preserve utility of processing, the noise introduced cannot be very large, so the filtering is also successful. On the other hand, the secret transformation keys $\mathcal{K}_{X,Y}$ used in ERB give authorized users additional information to correctly reconstruct the data, even if the magnitude of injected noise is large. In contrast, such information is not available to the attacker; therefore, noise filtering is not successful.

6.2 Effect of Data Distribution

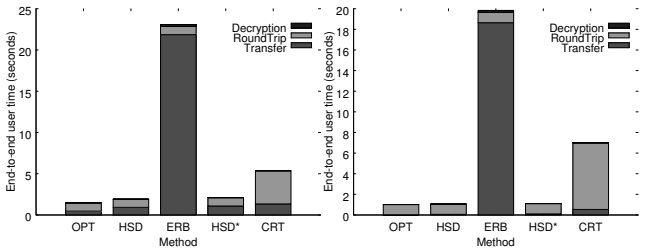
The next experiment studies the effect of data distribution on our transformation techniques. Table 5 shows the communication cost of range queries and kNN queries for different datasets. In general, the cost trends for both range and kNN queries are similar. The costs of our techniques (HSD, ERB, HSD*, CRT) are much lower than that of the baseline solution BULK. ERB is relatively expensive because the amount of query expansion depends on the error threshold ϵ. The HSD* solution shares the features of both HSD and ERB, yet its cost is much below that of ERB. CRT incurs the overhead of transferring the intermediate nodes; this overhead becomes especially high in the case of kNN queries. Observe that in the CRT method, the server is required to send at least one path of tree nodes to the client, so its communication cost for kNN queries is much higher than that of HSD/HSD*.

Figure 11 illustrates the end-to-end user time of the proposed methods evaluating range queries and kNN queries on the NE dataset. The *end-to-end user time* has three components: (*i*) the data transfer time, (*ii*) the round-trip network delay time, and (*iii*) the client-side decryption time. The client used in this experiment is a Pocket PC (HP iPAQ hw6915, with a 416MHz CPU). The AES decryption time and the SHA computation time for a data point are both 20 microseconds at the client side. The transfer bandwidth of GPRS is 80Kbits/second and the round-trip delay time

Table 5 Communication cost (KBytes) of queries, default setting

Dataset	Range query cost					
	BULK	OPT	HSD	ERB	HSD*	CRT
OL	119.23	0.17	0.30	9.05	0.36	3.77
TG	356.69	1.62	3.66	53.29	4.04	7.12
NE	2413.92	4.61	9.21	218.65	10.88	13.43
SF	3417.10	12.02	22.98	417.75	26.04	24.17
NA	3433.84	4.97	8.50	305.22	10.35	13.92
Dataset	**kNN query cost**					
	BULK	OPT	HSD	ERB	HSD*	CRT
OL	119.23	0.098	0.64	8.74	0.73	3.51
TG	356.69	0.098	1.69	47.19	1.91	3.79
NE	2413.92	0.098	0.60	186.44	1.06	5.36
SF	3417.10	0.098	2.90	353.22	3.70	5.44
NA	3433.84	0.098	0.37	254.65	0.90	4.85

is 1 second [31]. The cost of BULK is not shown here because it uses 241 seconds for transferring the data alone, not to mention its high client-side decryption time. The cost of OPT is shown for comparison purpose only. Observe that all of HSD, ERB, HSD*, and CRT incur only a low client-side decryption time. HSD* is slightly more expensive than HSD. In this experiment, CRT incurs an average of 4 round-trips per range query and 6.5 round-trips per kNN query. In comparison, HSD/HSD* only incurs a single round-trip per query. Thus, the end-to-end user time of CRT is substantially higher than that of HSD/HSD*, even though they have comparable data transfer time.



(a) range query cost (b) kNN query cost

Fig. 11 End-to-End User Time (Seconds), NE Data

We proceed to study the robustness of our methods against the general attack. Table 6 shows the attacker's estimation error for all datasets; the higher this value, the better. HSD* has an estimation error similar to that of HSD because the global distortion of the data in HSD* is determined by the key parameters in HSD, not by the value of ϵ. HSD/HSD* consistently leads to a higher estimation error than ERB, for all datasets.

We also considered the tailored attack on HSD, measuring the percentage of parameters of the transformation key that can be determined by the attacker (i.e., *leaking percentage*). We found that the leakage of HSD remains low (i.e., at most 5%) across all datasets, and we found that the method is insensitive to the data distribution.

Table 6 Attacker Estimation Error

Dataset	HSD	ERB	HSD*
OL	0.1186	0.0745	0.1181
TG	0.1940	0.0807	0.1933
NE	0.2045	0.0834	0.2041
SF	0.1342	0.0837	0.1347
NA	0.1254	0.0793	0.1250

In subsequent experiments, dataset NE is used as the default.

6.3 Effect of Target Set Distribution

Recall that HSD/HSD* employs a target dataset T for extracting part of the key parameters for transformation. Specifically, the original dataset P is transformed in such a way that its distribution becomes similar to that of T.

In the next experiment, we fix P to the NE dataset (see Figure 12a). Then, we study the impact of the estimation error, with respect to various distributions of T (see Figures 12b.1–f.1). The caption 'NE $\rightsquigarrow$ UI' of Figure 12b.2 corresponds to the transformed dataset obtained by fixing P to NE and setting T to UI. The other transformed datasets are shown in Figures 12c.2–f.2. Clearly, the transformed datasets appear very similar to their respective target dataset, but are different from the original dataset. We then apply the general attack (with the default setting) on these transformed datasets, and obtain their corresponding estimated datasets in Figures 12b.3–f.3. The attacker only learns that most of the points are distributed on the left side of the map; however, the data distributions of these estimated datasets remain significantly different from the original dataset P. Specifically, the estimation error of each estimated dataset from the original dataset is listed in Table 7. Depending on the target data distribution, the estimation error ranges from 0.189 to 0.275. Although the NE dataset (original) looks similar to the SF dataset (target), the estimation error remains acceptable. Observe that the estimated dataset in Figure 12f.3 appears very different from the original dataset in Figure 12a.

Table 7 Attacker Estimation Error, P = NE dataset

Target set T	HSD	HSD*
UI	0.2045	0.2041
OL	0.2536	0.2535
TG	0.2757	0.2743
SF	0.1903	0.1896
NA	0.2273	0.2277

We proceed to study the communication cost of range and kNN queries on the above transformed datasets. Table 8 shows the query cost for various target sets T. For range queries, the cost of HSD/HSD* is relatively insensitive to the distribution of the target set, and HSD* is only 20% more

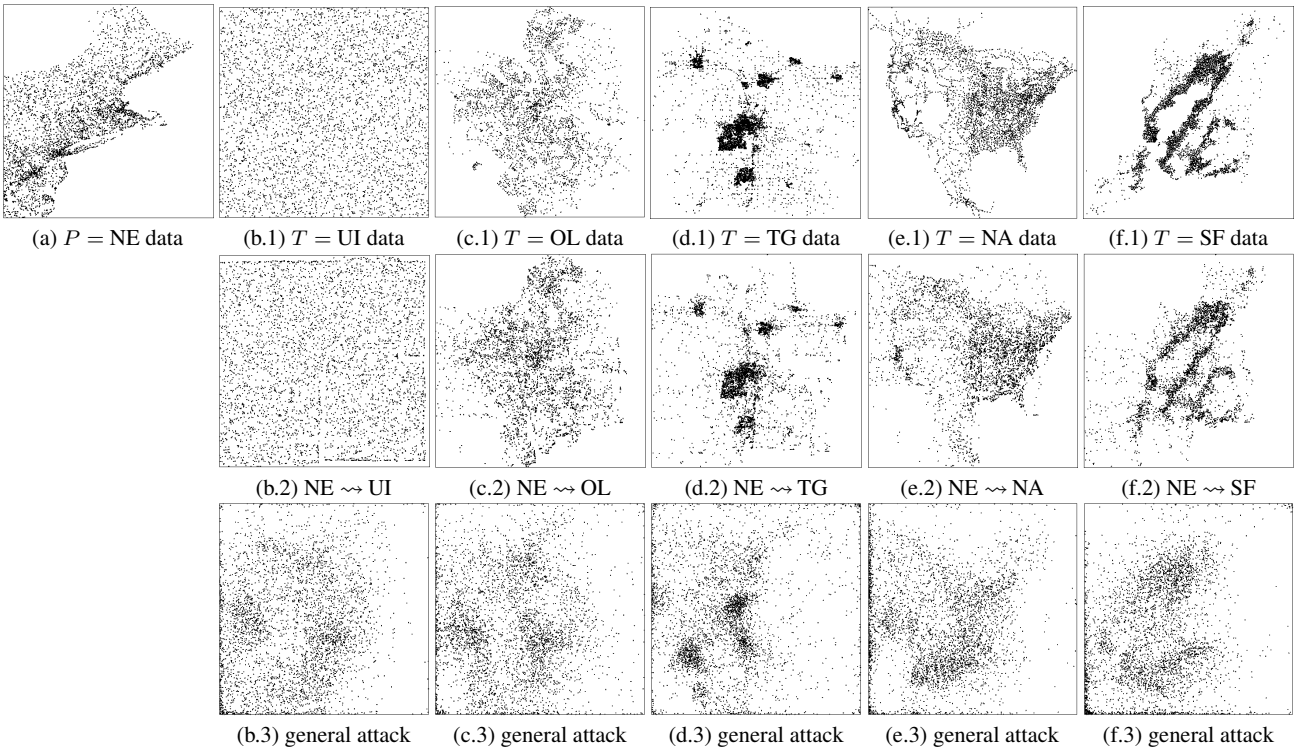


Fig. 12 Effect of the Target Set on the Transformed Dataset by HSD/HSD*

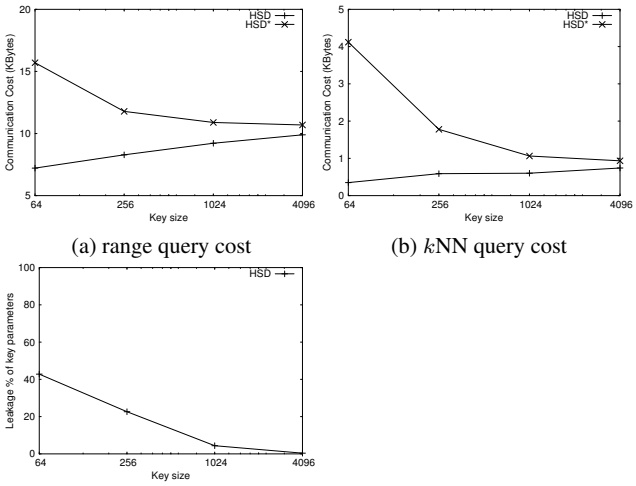
expensive than HSD. The cost of kNN queries is more sensitive to the distribution of T. Interestingly, the target set TG that incurs the highest query cost for kNN queries is also the one that leads to the highest estimation error (see Table 7).

Table 8 Communication cost (KBytes), P = NE dataset

Target set T	**Range query cost**		k**NN query cost**	
	HSD	HSD*	HSD	HSD*
UI	9.217	10.889	0.602	1.063
OL	11.616	13.485	2.006	2.692
TG	10.580	12.417	5.739	10.132
SF	10.376	12.119	2.478	4.266
NA	11.501	13.379	0.918	1.802

6.4 Effect of Method-Specific Parameters

Figure 13a,b illustrate the query cost of HSD/HSD* for range queries and kNN queries, respectively, for different values of the key size Υ. The cost of HSD rises slightly as Υ increases. At a large Υ value, the area of each partition becomes small, so the relative amount of noise injected by HSD* into the dataset also becomes small. This explains why the cost of HSD* decreases when Υ increases. We then investigate the tailored attack on HSD—Figure 13c illustrates the leakage percentage of HSD as a function of Υ. The leakage percentage drops quickly as Υ increases. It is practically safe to use a moderate key size (e.g., 1024) for HSD.



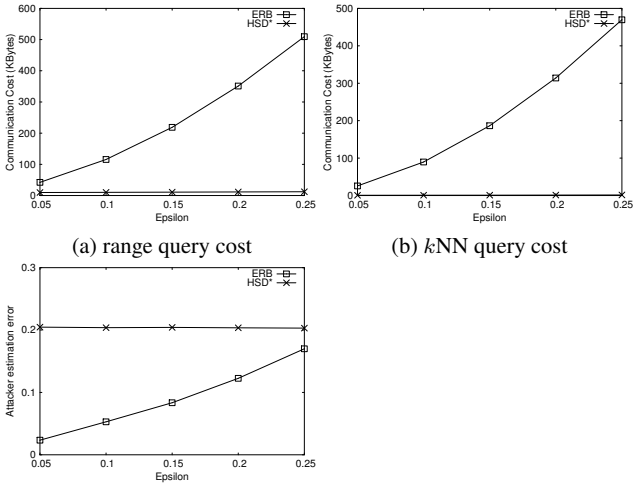
(a) range query cost (b) kNN query cost

(c) leakage percentage

Fig. 13 Effect of the Key Size Υ on HSD/HSD*

Figures 14a,b show the cost of ERB/HSD* for range and kNN queries as a function of ϵ. Observe that the cost of HSD* is much lower than that of ERB and that the performance gap widens as ϵ increases. Next, we study the impact of the general attack on ERB/HSD*; Figure 14c plots the estimation error with respect to ϵ. ERB provides the data owner flexibility in tuning the attacker's estimation error (by changing the ϵ value). The estimation error of HSD* is not significantly influenced by ϵ. The reason is that the rough outline of the transformation is defined by the parameters of

the spatial transformation key; the value ϵ is only used to displace the points within the same partition.

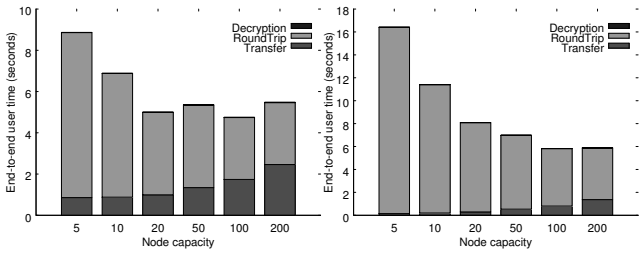


(a) range query cost (b) kNN query cost

(c) estimation error

Fig. 14 Effect of the Error Threshold ϵ on ERB/HSD*

Figures 15a,b show the end-to-end user time of CRT for range and kNN queries when varying the node capacity of the encrypted R*-tree. At a small node capacity value, the tree has many levels, so the majority of time is spent on the round-trips. On the other hand, the data transfer time becomes high at a large node capacity. The cost of CRT follows a 'U'-shaped trend, which is consistent with the prediction of our cost model from Section 5.2. Since the optimal capacity stays close to 50, we fix the node capacity value to 50 in the remaining experiments.

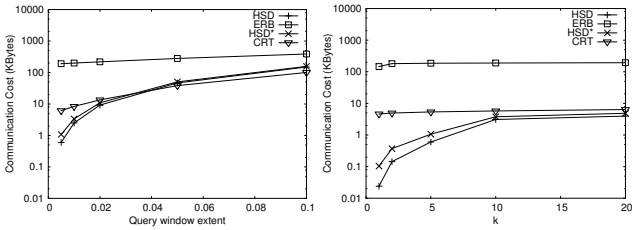


(a) range query cost (b) kNN query cost

Fig. 15 Effect of Node Capacity on End-to-End User Time (Seconds)

6.5 Effect of Query Parameters

Figure 16a,b plot the communication cost of range and kNN queries as a function of the query extent and k, respectively. ERB incurs a much higher cost when compared to its competitors. Due to the overhead of intermediate nodes, CRT is more expensive than HSD/HSD* for small query extents or small k. However, for large query extents or large k, the overhead of CRT becomes relatively small, and its cost becomes close to that of HSD/HSD*.



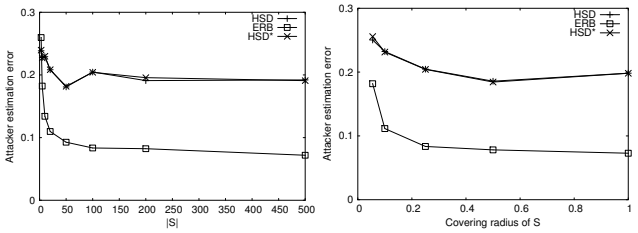
(a) varying range extent (b) varying k

Fig. 16 Effect of Query Parameters on Query Communication Cost

6.6 Effect of the known set S

We proceed to study the impact of the general attack on HSD/ERB/HSD*. Figure 17a plots the attacker estimation error as a function of the number $|S|$ of known points. When $|S|$ increases, the estimation error decreases and then converges to a constant value. The convergence occurs because, for a larger S (and a fixed covering radius r), the known points tend to cluster together, thus carrying redundant information. Hence, the attacker gains no additional knowledge. The protection of HSD/HSD* is better than that of ERB.

Figure 17b shows the estimation error with respect to the covering radius r of S. Compared to ERB, HSD/HSD* is more powerful in distorting the space of transformed points. When r is low, all points in S are close to each other, and little information is leaked to the attacker. As r grows, the known points are more uniformly distributed in the original space, increasing the success probability of an attack (i.e., lower estimation error). The error for HSD/HSD* stabilizes after the covering radius reaches 50% of the space, thus confirming its robustness.



(a) vs. $|S|$ (b) vs. covering radius of S

Fig. 17 Estimation Error with respect to S

7 Conclusion

Content sharing and collaboration services allow subscribers to share private spatial data (e.g., points of interest, geo-tagged business data) with authorized users. In this scenario, it is attractive to be able to maintain data confidentiality with respect to untrusted parties, including the service provider. The paper presents methods to encode a dataset such that

only authorized users can access the content, while the service provider "blindly" evaluates queries, without seeing the actual data. We contribute a spatial transformation HSD*, that allows efficient spatial query processing. We study its security guarantees under tailored and general attack models. We also contribute the CRT method, which is based on encryption and offers very high confidentiality guarantees, but also incurs a large number of communication rounds.

It is important for the data owner to choose an appropriate transformation method that best matches her requirements. Recall that our transformation methods achieve different trade-offs between data privacy and query efficiency. In case the data owner requires perfect data privacy, we recommend the CRT method, whose end-to-end user time is only three times that of the optimal method (OPT). For other cases, we recommend the HSD* method because it offers the advantages of both of the preliminary methods HSD and ERB: (i) efficient query processing, (ii) robustness against the general attack, and (iii) robustness against the tailored attack when the adversary has polynomially bounded computational power.

A promising future direction is to extend the proposed techniques to also support other types of spatial queries in addition to the fundamental range and kNN queries, including spatial joins and skyline queries.

References

1. *National Institute of Standards and Technology. Secure Hashing. http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html.*
2. Advanced Encryption Standard (AES). NIST - Federal Information Processing Standards Publication 197, Nov 2001.
3. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-Preserving Encryption for Numeric Data. In *SIGMOD*, 2004.
4. R. Agrawal and R. Srikant. Privacy-Preserving Data Mining. In *SIGMOD*, 2000.
5. C. Böhm. A Cost Model for Query Processing in High-Dimensional Data Spaces. *ACM TODS*, 25(2):129–178, 2000.
6. T. Brinkhoff. A Framework for Generating Network-based Moving Objects. *GeoInformatica*, 6(2):153–180, 2002.
7. A. R. Butz. Alternative Algorithm for Hilbert's Space-Filling Curve. *IEEE Trans. Comput.*, C-20(4):424–426, 1971.
8. W. Cheng, H. Pang, and K.-L. Tan. Authenticating Multi-dimensional Query Results in Data Publishing. In *DBSec*, 2006.
9. E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs. In *CCS*, 2003.
10. P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic Data Publication over the Internet. *J. Comput. Secur.*, 11(3):291–314, 2003.
11. C. Dwork. Differential privacy: A survey of results. In *TAMC*, pages 1–19, 2008.
12. B. Gedik and L. Liu. Location Privacy in Mobile Systems: A Personalized Anonymization Model. In *ICDCS*, 2005.
13. G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K. L. Tan. Private Queries in Location Based Services: Anonymizers are not Necessary. In *SIGMOD*, 2008.
14. G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis. Fast Data Anonymization with Low Information Loss. In *VLDB*, 2007.
15. O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM*, 43:431–473, 1996.
16. M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *USENIX MobiSys*, 2003.
17. H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over Encrypted Data in the Database-Service-Provider Model. In *SIGMOD*, 2002.
18. H. Hacigümüs, S. Mehrotra, and B. R. Iyer. Providing Database as a Service. In *ICDE*, 2002.
19. G. R. Hjaltason and H. Samet. Distance Browsing in Spatial Databases. *TODS*, 24(2):265–318, 1999.
20. P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preventing Location-Based Identity Inference in Anonymous Spatial Queries. *IEEE TKDE*, 19(12):1719–1733, 2007.
21. H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the Privacy Preserving Properties of Random Data Perturbation Techniques. In *ICDM*, 2003.
22. A. Khoshgozaran and C. Shahabi. Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy. In *SSTD*, 2007.
23. K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian Multidimensional K-Anonymity. In *ICDE*, 2006.
24. N. Li, T. Li, and S. Venkatasubramanian. t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In *ICDE*, 2007.
25. K. Liu, H. Kargupta, and J. Ryan. Random Projection-Based Multiplicative Data Perturbation for Privacy Preserving Distributed Data Mining. *IEEE TKDE*, 18(1):92–106, 2006.
26. A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-Diversity: Privacy Beyond k-Anonymity. In *ICDE*, 2006.
27. R. C. Merkle. A Certified Digital Signature. In *CRYPTO*, 1989.
28. M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The New Casper: Query Processing for Location Services without Compromising Privacy. In *VLDB*, 2006.
29. S. Papadimitriou, F. Li, G. Kollios, and P. S. Yu. Time Series Compressibility and Privacy. In *VLDB*, 2007.
30. P. Samarati. Protecting Respondents' Identities in Microdata Release. *IEEE TKDE*, 13(6):1010–1027, 2001.
31. P. Stuckmann, N. Ehlers, and B. Wouters. GPRS Traffic Performance Measurements. In *IEEE Vehicular Technology Conference*, 2002.
32. Y. Theodoridis and T. K. Sellis. A Model for the Prediction of R-tree Performance. In *PODS*, 1996.
33. R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB*, 1998.
34. P. Williams, R. Sion, and B. Carbunar. Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, pages 139–148, 2008.
35. W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis. Secure k-NN Computation on Encrypted Databases. In *SIGMOD*, 2009.
36. Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios. Spatial Outsourcing for Location-based Services. In *ICDE*, 2008.
37. M. L. Yiu, G. Ghinita, C. S. Jensen, and P. Kalnis. Outsourcing Search Services on Private Spatial Data. In *ICDE*, 2009.
38. M. L. Yiu, C. S. Jensen, X. Huang, and H. Lu. SpaceTwist: Managing the Trade-Offs Among Location Privacy, Query Performance, and Query Accuracy in Mobile Services. In *ICDE*, 2008.