Secure and Efficient Range Queries on Outsourced Databases Using $\widehat{\mathrm{R}}$-trees

Peng Wang and Chinya V. Ravishankar

Department of Computer Science & Engineering, University of California–Riverside, Riverside, CA 92521, USA
{wangpe, ravi}@cs.ucr.edu

***Abstract*—We show how to execute range queries securely and efficiently on encrypted databases in the cloud. Current methods provide either security or efficiency, but not both. Many schemes even reveal the ordering of encrypted tuples, which, as we show, allows adversaries to estimate plaintext values accurately.**

We present the $\widehat{\mathrm{R}}$-tree, a hierarchical encrypted index that may be securely placed in the cloud, and searched efficiently. It is based on a mechanism we design for encrypted halfspace range queries in $\mathbb{R}^d$, using Asymmetric Scalar-product Preserving Encryption.

Data owners can tune the $\widehat{\mathrm{R}}$-tree parameters to achieve desired security-efficiency tradeoffs. We also present extensive experiments to evaluate $\widehat{\mathrm{R}}$-tree performance. Our results show that $\widehat{\mathrm{R}}$-tree queries are efficient on encrypted databases, and reveal far less information than competing methods.

I. Introduction

The term cloud computing refers to a broad range of outsourcing services for storage and computation [1]. This model is growing in popularity because users have the appearance of virtually unbounded resources, but more significantly, because they are relieved of the burden of managing these resources. Outsourcing of large databases has therefore become a well-studied topic.

However, this model has its costs. Outsourced data must be encrypted to preserve its privacy and integrity, but encryption makes queries harder to run. Conventional encryption schemes, such as block ciphers [2], do not directly support the sorts of comparisons, searches, and other manipulations needed to handle queries without loss of privacy. New encryption schemes [3]–[9] have hence been proposed to facilitate queries on encrypted data.

Security and efficiency are both important considerations when designing such encryption schemes. Some schemes [4], [5] achieve efficiency at the cost of revealing the relative order of encrypted data points. We will show that this is dangerous; the adversary can exploit ordering information to estimate data points precisely, using order statistics [10].

Predicate-encryption based query schemes (PREs) [4], [6], [7], [11], [12] offer provable security for encryption, but suffer from high computation overhead. The overhead of range queries in these schemes increases significantly with the query range or the precision required.

Some ordering information leakage is likely inevitable, but the challenge is to minimize such leakage. For instance, bucketization schemes [9], [13] make a tradeoff between ordering information leakage and efficiency. Data is grouped into buckets, and indices are built on buckets, rather than on data tuples. A query response includes all the tuples in all matching buckets, and may hence contain false positives. A query now leaks no ordering information inside each bucket, since the adversary cannot distinguish tuples within a bucket.

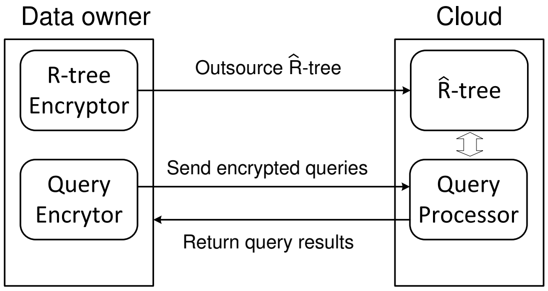


Fig. 1: Scheme model.

However, current bucketization schemes suffer from various limitations. For instance, bucket indices must be stored and searched locally at the data owner's site, rather than at the cloud. Also, bucket indices are not hierarchical, so searching them requires time linear in the number of buckets.

The Asymmetric Scalar-product Preserving Encryption (ASPE) approach [14] allows relative distance comparison between data points under encryption. Given a query point Q and any two data points P_1, P_2, all encrypted, ASPE permits one to determine whether Q is closer to P_1 or P_2. It has been used for secure k-Nearest-Neighbor (kNN) computation on encrypted data.

A. Our Contributions

We make the following contributions in this paper. First, we propose an innovative method for running encrypted halfspace range queries in $\mathbb{R}^d$ on points encrypted with ASPE. This mechanism can achieve polyhedral queries on encrypted data.

Using this mechanism, we present the $\widehat{\mathrm{R}}$-tree, an indexing scheme for encrypted and outsourced databases. The $\widehat{\mathrm{R}}$-tree uses ASPE to encrypt the query ranges; the data itself can be encrypted any other way. The $\widehat{\mathrm{R}}$-tree is a hierarchical bucketization scheme, but in contrast to current bucketization schemes, encrypted $\widehat{\mathrm{R}}$-tree indices are stored and queried entirely in the cloud, rather than at the data owner site. $\widehat{\mathrm{R}}$-trees permit us to outsource data management more effectively.

The $\widehat{\mathrm{R}}$-tree is similar to an R-tree in structure. The minimum bounding box (MBR) of each R-tree node is treated as a

978-1-4673-4910-9/13/$31.00 © 2013 IEEE

bucket, and encrypted using ASPE. Parent-children relationships are preserved, so queries can be performed recursively. If a query region overlaps a leaf's MBR in the $\widehat{\mathrm{R}}$-tree, all data points in the MBR are returned. This reduces ordering information leakage, but may introduce false positives, a common tradeoff in bucketization schemes.

We also present a theoretical analysis of the risks arising from leakage of ordering information. We will show that when ordering information is available, the adversary can, using simple mechanisms, estimate the values of data points much more precisely than when ordering information is unavailable.

We show that $\widehat{\mathrm{R}}$-trees allow data owners to make tradeoffs between ordering information leakage and efficiency by using parameters such as node fanout to tune the tree structure. Data owners can tune the tree structure according to their security and efficiency needs.

Finally, we present an $\widehat{\mathrm{R}}$-tree implementation, and compare its performance with that of regular R-trees through extensive experiments. We also compare $\widehat{\mathrm{R}}$-tree with PREs like [4]. Our results show that $\widehat{\mathrm{R}}$-tree algorithms are 1200 to 25000 times faster than those in PREs, and $\widehat{\mathrm{R}}$-tree query time is even comparable with that in ordinary R-trees. This is excellent, considering that we perform encrypted queries on encrypted indices.

The rest of the paper is organized as follows. Related work appears in Sec. II. Sec. III introduces preliminaries, including ASPE. Sec. IV gives an overview of our scheme and the security model. Sec. V presents our scheme. Sec. VI analyzes the dangers of revealing ordering information, and Sec. VII presents experiments on real-world data sets comparing the resilience of our scheme and schemes that reveal ordering information. Sec. VIII presents a theoretical analysis of $\widehat{\mathrm{R}}$-tree's performance. Sec. IX presents the $\widehat{\mathrm{R}}$-tree implementation, and its performance. Sec. X concludes the paper.

II. Related Work

The issue of secure outsourcing of data has been addressed in several recent papers. In [14], Wong et al. propose a scheme for secure kNN queries on encrypted data. Distance comparisons between an encrypted query and data points are achieved using ASPE, with query points and data points being encrypted differently. Encrypted data points carry information relating to their distance to the origin. Given two data points and a query point, the cloud can determine which data point is closer to the query point. Using artificial dimensions and randomly splitting, ASPE is proved secure against known-plaintext attacks.

In [9], Hore et al. partition the data into a set of buckets. The data owner builds indices for buckets, outsources all data to the cloud, but retains the indices at his site. The bucket ranges are not hierarchically structured, so the index search must be linear in the number of buckets. To process a range query, the data owner finds the set of buckets intersecting the query range, and retrieves these buckets from the cloud. Increasing bucket size improves privacy, but also increases the false positive rate. However, if we make buckets smaller to reduce false positive rate, we also increase their number. This reduces efficiency, since the indices must be locally stored, and index searching is linear in the number of buckets. Such drawbacks restrict the application of this scheme.

In [5], Boldyreva et al. proposed the Order-Preserving Symmetric Encryption (OPSE). If $E(p)$ represents the encryption of plaintext p, then $p_1 > p_2$ guarantees $E(p_1) > E(p_2)$ in OPSE. One can hence construct indices and run efficient range queries on encrypted data. In [15], the authors revisited the security of the scheme, and improve its security. Unfortunately, this scheme is not practical, since it reveals the ordering of encrypted tuples, which can lead to substantial privacy loss.

In [4], Lu proposed an outsourced range query scheme using predicate encryption [11]. This scheme provides provable security for outsourced data and queries, and can achieve logarithmic-time search since it orders the encrypted data points. However, it is not very practical. It only supports one-dimensional data points. Most damaging is that this method reveals ordering information, which can lead to unacceptable privacy loss, as we will show.

The R-tree [16] is a height-balanced tree used for indexing multi-dimensional data. Each R-tree node contains several entries. Each leaf node entry has the form $(\textit{object-id}, R)$ where *object-id* is the object's identifier and R is the MBR of the data object. Each internal node entry is of the form $(\textit{ptr}, R)$ where *prt* is a pointer to a lower level node and R is its MBR. R-trees support efficient range queries. R-tree variants, such as the R^+-tree and the R^*-tree, incorporate various enhancements.

III. Preliminaries

We briefly review several ideas central to our presentation.

A. Asymmetric Scalar-product-Preserving Encryption

Asymmetric Scalar-product-Preserving Encryption (ASPE) was proposed in [14], for performing kNN queries on encrypted data points in $\mathbb{R}^d$. Encryption uses a $(d+1) \times (d+1)$ invertible matrix M as the secret key. Data and queries are encrypted differently, a difference we recognize in our notation.

Point_Enc$(P, M) \rightarrow \langle P \rangle$. This function accepts a data point $P \in \mathbb{R}^d$ and a $(d+1) \times (d+1)$ key matrix M, and outputs the ciphertext $\langle P \rangle$ of P. It first creates a point $P_+ \in \mathbb{R}^{d+1}$, such that $P_+ = (P^T | (-0.5\|P\|^2))^T$, where $\|P\|$ is the Euclidean norm of P. The encryption of P is $\langle P \rangle = M^T P_+$.

Query_Enc$(Q, M^{-1}) \rightarrow [Q]$. This function accepts a query point $Q \in \mathbb{R}^d$ and M^{-1}, the inverse of the key matrix M. It outputs $[Q]$, the ciphertext of Q. It first creates a point $Q_+ \in \mathbb{R}^{d+1}$, such that $Q_+ = r(Q^T|1)^T$, where r is a random positive number. The encrypted point $[Q] = M^{-1}Q_+$.

Dist_Comp$(\langle P \rangle, \langle P' \rangle, [Q])) \rightarrow \{0, 1\}$. This function accepts two encrypted data points $\langle P \rangle$, $\langle P' \rangle$, an encrypted query point $[Q]$, and returns 1 iff P is closer to Q than P'. It outputs the Boolean value $(\langle P\rangle - \langle P'\rangle) \cdot [Q] > 0$. Now,

$$\begin{aligned}(\langle P\rangle - \langle P'\rangle) \cdot [Q] &= (\langle P\rangle - \langle P'\rangle)^T [Q] \\ &= (M^T (P_+ - P'_+))^T M^{-1} Q_+ \\ &= (P_+ - P'_+)^T Q_+ \\ &= (P - P')^T (rQ) + r(-0.5\|P\|^2 + 0.5\|P'\|^2) \\ &= 0.5r(\|P' - Q\| - \|P - Q\|),\end{aligned}$$

where $\|P - Q\|$ is the Euclidean distance between P and Q. This expression is positive iff P is closer to Q than is P'. A kNN query identifies the k nearest points by comparing the distance from the query point Q to each data point P.

B. Halfspace Range Queries

Halfspace range queries (hRQ) are a fundamental problem in computational geometry, since any form of algebraic range searching can be transformed into it [17]. If $\mathbf{a} \in \mathbb{R}^d, \mathbf{a} \neq \mathbf{0}$, and $b \in \mathbb{R}$, a *hyperplane* $\mathcal{H}$ is defined by the set $\mathbf{x} \in \mathbb{R}^d$ such that $\mathbf{a}^T\mathbf{x} = b$ [18]. $\mathcal{H}$ partitions $\mathbb{R}^d$ into the *inner halfspace* $\mathcal{H}^{\leqslant}$ corresponding to $\mathbf{a}^T\mathbf{x} \leqslant b$, and the *outer halfspace* $\mathcal{H}^{>}$ corresponding to $\mathbf{a}^T\mathbf{x} > b$. Every $\mathcal{S} \subseteq \mathbb{R}^d$ is partitioned by $\mathcal{H}$ into two disjoint subsets $\mathcal{S}^{\leqslant}_{\mathcal{H}} = \mathcal{S} \cap \mathcal{H}^{\leqslant}$ and $\mathcal{S}^{>}_{\mathcal{H}} = \mathcal{S} \cap \mathcal{H}^{>}$.

Given a set of points $\mathcal{S} = \{P_1, P_2, \ldots, P_n\}$ and a hyperplane $\mathcal{H}$ in $\mathbb{R}^d$, a *halfspace range query* asks for $\mathcal{S}^{\leqslant}_{\mathcal{H}}$.

C. Order Statistics

Order statistics are an important tool in non-parametric statistics [10]. Let $X_1, X_2, \ldots, X_n$ be i.i.d. random variables with the density and distribution functions $f(x)$ and $F(x)$, respectively. Let the X_i be sorted to get $X_{(1)} \leqslant X_{(2)} \leqslant \ldots, \leqslant X_{(n)}$. Now, $X_{(k)}$ is called the kth *order statistic*. It can be shown that the density function of $X_{(k)}$ is given by [5]:

$$f_{X_{(k)}}(x) = \binom{n}{1}\binom{n-1}{k-1} f(x)[F(x)]^{k-1}[1 - F(x)]^{n-k} \tag{1}$$

IV. Overview

We now introduce our system and security model, and provide an overview of $\widehat{\text{R}}$-tree. Our approach, unlike [14], uses an index to speed up queries. Also, we decouple the encryption of data points from that of queries and the index.

A. System Model

Our model recognizes two entities: the data owner and the cloud service provider. (see Fig. 1). The data owner places encrypted data and a corresponding $\widehat{\text{R}}$-tree index in the cloud, which provides infrastructure for computing and storage. The data owner creates and sends encrypted queries for ranges of interest to the cloud. The cloud performs encrypted queries on the $\widehat{\text{R}}$-tree index, and returns query results to the data owner.

The data owner creates an $\widehat{\text{R}}$-tree by first building a regular R-tree for the given set of points $\mathcal{S} \subseteq \mathbb{R}^d$. The MBR ranges are encrypted using ASPE to obtain the $\widehat{\text{R}}$-tree. The parent-children relationships in the $\widehat{\text{R}}$-tree are not encrypted.

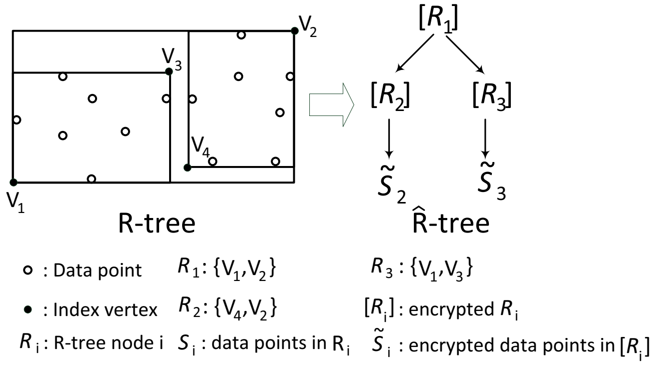


Fig. 2: R-tree and $\widehat{\text{R}}$-tree.

While the R-tree MBR ranges are encrypted using ASPE to support range queries, the data points in $\mathcal{S}$ may be encrypted independently, by other encryption schemes, such as block ciphers. A d-dimensional range R can be defined by its two extremal vertices. The R-tree in Fig. 2 contains nodes with MBRs $\mathcal{R}_1 = (V_1, V_2)$, $\mathcal{R}_2 = (V_4, V_2)$, and $\mathcal{R}_3 = (V_1, V_3)$. The data sets contained in $\mathcal{R}_2$ and $\mathcal{R}_3$ are S_2 and S_3, respectively.

MBR ranges in the $\widehat{\text{R}}$-tree are encrypted by applying ASPE to each extremal vertex used to define a range. As shown in Fig. 2, the corresponding $\widehat{\text{R}}$-tree contains three nodes $[\mathcal{R}_1] = ([V_1], [V_2])$, $[\mathcal{R}_2] = ([V_4], [V_2])$, $[\mathcal{R}_3] = ([V_1], [V_3])$. Data points inside each leaf MBR are encrypted using a conventional encryption scheme. The encrypted versions of S_2 and S_3 are $\widetilde{S}_2$ and $\widetilde{S}_3$.

The data owner creates encrypted range queries, and sends them to the cloud. The cloud searches the $\widehat{\text{R}}$-tree, performing intersection tests level-by-level, as in an ordinary R-tree, and proceeding to a node's children if and only if the node's bounding box intersects the query range. The cloud thereby obtains all leaves intersecting the query range, and he returns encrypted data points in these leaves to the data owner. The cloud cannot query the data points themselves, since they are encrypted separately.

The $\widehat{\text{R}}$-tree may introduce false positives in query results, but protects ordering information inside each leaf MBR, a reasonable tradeoff. Precise query schemes [4], [12], which return exactly the set of encrypted tuples in the query range, can leak information over time. Given enough range query results, the adversary can reconstruct the ordering of tuples from unions and intersections of these result clusters.

B. Security Model

We adopt a "honest but curious" model for our adversary, the cloud server. Its goal is to learn the plaintexts for encrypted data. It may know some knowledge of the outsourced data set, and try to use this knowledge to obtain the values of points in the data set. Otherwise, it is scrupulous in following the protocol defined by the data owner, and will return the right query results.

ASPE, which we use to encrypt index ranges and queries, is secure against known-plaintext attacks [14]. The adversary, however, may mount more complex attacks. For instance, it can gain some information about the ordering of encrypted data values while processing queries. The adversary may know the distribution of plaintext values of all data points, and some data points' values. It will try to estimate the values of other data points using such information.

1) Attacks Based On Order Statistics: Let us begin by assuming that the adversary knows the ordering of the encrypted data points. Some encryption schemes, such as [4], [5], reveal this ordering explicitly. In other cases, it may be possible to infer this ordering over time from queries. It is also often possible to obtain the distributions of the data values either from public sources, or by examining other available and similar data sets.

Using such knowledge of distributions and ordering, the adversary can use order statistics methods to estimate the plaintext values for the encrypted tuples. For one-dimensional data, say that the adversary learns the plaintext values of m data points $y_{i_1} < y_{i_2} < \ldots < y_{i_m}$. It uses these points as the endpoints to obtain $m-1$ ranges $[y_{i_1}, y_{i_2}], \ldots, [y_{i_{m-1}}, y_{i_m}]$. It knows the ordering of encrypted tuples in each range, and can now arrive at better estimates for plaintext values of encrypted tuples in each range using order statistics. For multidimensional data, the adversary can perform the same attack to better estimate values of encrypted tuples on each dimension.

$\widehat{\text{R}}$-trees do not reveal the full ordering of data points, but do leak information on the ordering of leaf MBRs. We will study the effectiveness of attacks outlined above on $\widehat{\text{R}}$-trees.

C. Halfspace Range Query on Encrypted Data

Queries in [14] ask which encrypted data points in $\mathbb{R}^d$ are closest to a given encrypted query point. Their method converts each data point in $\mathbb{R}^d$ to a point in $\mathbb{R}^{d+1}$, the additional dimension encoding the point's distance from the origin. However, query points are not required to carry such distance information. Our approach to encrypted halfspace range queries (EhQ) is the dual of this method, and must check which of two query points is closer to a vertex in an $\widehat{\text{R}}$-tree MBR. Hence query points are embedded with distance information in our scheme, while points corresponding to MBR vertices are not.

We construct halfspace range queries as in Fig. 3. Given a hyperplane $\mathcal{H}$ and the corresponding halfspaces $\mathcal{H}^{\leqslant}$ and $\mathcal{H}^{>}$, we select *anchor points* $\boldsymbol{\omega}^{\leqslant} \in \mathcal{H}^{\leqslant}$ and $\boldsymbol{\omega}^{>} \in \mathcal{H}^{>}$ equidistant from $\mathcal{H}$, such that the line segment $(\boldsymbol{\omega}^{>}, \boldsymbol{\omega}^{\leqslant})$ is normal to $\mathcal{H}$. Every point on $\mathcal{H}$ is now equidistant from $\boldsymbol{\omega}^{\leqslant}$ and $\boldsymbol{\omega}^{>}$, but points in $\mathcal{H}^{\leqslant}$ are closer to $\boldsymbol{\omega}^{\leqslant}$ and points in $\mathcal{H}^{>}$ are closer to $\boldsymbol{\omega}^{>}$. We can check whether a given point V is in $\mathcal{H}^{\leqslant}$ or $\mathcal{H}^{>}$ by checking whether V is closer to $\boldsymbol{\omega}^{\leqslant}$ or $\boldsymbol{\omega}^{>}$, just as in ASPE.

D. Index Searches as Hyperectangle Intersections

Searching $\widehat{\text{R}}$-trees requires us to determine whether the d-dimensional *query hyperrectangle* intersects the *index hyperrectangle* in an $\widehat{\text{R}}$-tree node. We make the usual assumptions that coordinate axes are orthogonal, and that each hyperrectangle face is orthogonal to some axis.

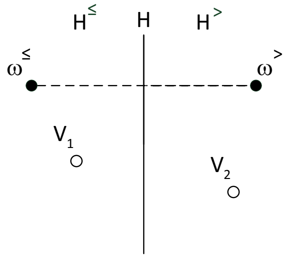


Fig. 3: Halfspace range query.

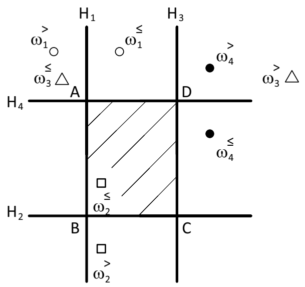
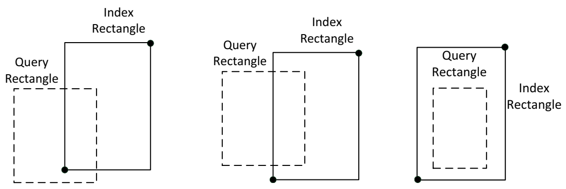


Fig. 4: Query rectangle $ABCD$.



(a) Case 1

(b) Case 2

(c) Case 3

Fig. 5: Three cases of rectangle intersection.

Our method is based on the observation that a d-dimensional query hyperrectangle $\mathcal{Q}$ can be defined as the space enclosed by the hyperplanes $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_{2d}$ defined by its $2d$ faces. We adopt the convention that the query region is defined to lie in $\mathcal{H}_i^{\leqslant}$ for each i. That is, the hyperplanes are so specified that the points $\mathbf{x}$ of interest satisfy $\mathbf{a}_i^T \mathbf{x} \leqslant b_i$. Under these conditions, we will have $\mathcal{Q} = \mathcal{H}_1^{\leqslant} \cap \mathcal{H}_2^{\leqslant} \cap \cdots \cap \mathcal{H}_{2d}^{\leqslant}$. Fig. 4 shows a 2-dimensional query rectangle defined by four halfspace range queries, or eight anchor points. The halfspaces $\mathcal{H}_i^{\leqslant}$ and $\mathcal{H}_i^{>}$ are defined by two anchor points $\boldsymbol{\omega}_i^{\leqslant}$ and $\boldsymbol{\omega}_i^{>}$. We select $\boldsymbol{\omega}_i^{\leqslant}$ at random in $\mathcal{H}_i^{\leqslant} - \mathcal{H}_i$. $\boldsymbol{\omega}_i^{>}$ will be its reflection in the hyperplane $\mathcal{H}_i$. We specify each index hyperrectangle $\mathcal{R} \subseteq \mathbb{R}^d$ in a $\widehat{\text{R}}$-tree node by its 2 vertices, as shown in Fig. 2.

Fig. 5 shows three cases of 2-dimensional rectangle intersections. Clearly, we cannot test for rectangle intersection merely by testing whether a vertices of one is included in the other. Instead, we must test whether or not the vertices are in the appropriate halfspaces defined by the faces of query $\mathcal{Q}$.

V. Our Scheme

Our approach to EhQ works as follows. To encrypt a query $\mathcal{Q} = \mathcal{H}_1^{\leqslant} \cap \mathcal{H}_2^{\leqslant} \cap \cdots \cap \mathcal{H}_{2d}^{\leqslant}$, we generate anchors $\boldsymbol{\omega}_i^{\leqslant}$ and $\boldsymbol{\omega}_i^{>}$ for each hyperplane $\mathcal{H}_i$. Then we generate an encrypted *discriminator* $\Delta_{\mathcal{H}_i}$ for each $\mathcal{H}_i$. Using $\Delta_{\mathcal{H}_i}$, we are able to determine *under encryption*, whether a given encrypted point V lies in $\mathcal{H}_i^{\leqslant}$ or in $\mathcal{H}_i^{>}$.

A. Vertex and Query Range Encryption Algorithms

Our method uses the following algorithms.

Enc_Vertex$(V, M) \rightarrow [V]$. The data owner uses this algorithm to encrypt a vertex V of the MBR of an $\widehat{\text{R}}$-tree node using his secret key M, an invertible $(d+1) \times (d+1)$ matrix. Given vertex $V = (v_1, v_2, \ldots, v_d)^T$, the algorithm first adds an additional dimension, to create $V_+ = (V^T|1)^T$. Vertex V is encrypted to $[V] = M^{-1}V_+$.

$\mathbf{Gen\_Anchor}(\mathcal{H}) \rightarrow (\boldsymbol{\omega}^{\leqslant}, \boldsymbol{\omega}^{>})$. This algorithm accepts a hyperplane $\mathcal{H}$ defined by parameters $\mathbf{a}$ and b, and outputs anchor points $\boldsymbol{\omega}^{\leqslant}$ and $\boldsymbol{\omega}^{>}$ lying in $\mathcal{H}^{\leqslant}$ and $\mathcal{H}^{>}$, respectively. It randomly selects a point $\boldsymbol{\omega}^{\leqslant} \in \mathcal{H}^{\leqslant} - \mathcal{H}$, and computes $\boldsymbol{\omega}^{>}$ as its reflection in $\mathcal{H}$, as follows. If $\boldsymbol{\omega}^{\leqslant}$ and $\boldsymbol{\omega}^{>}$ are vectors representing $\boldsymbol{\omega}^{\leqslant}$ and $\boldsymbol{\omega}^{>}$ respectively, we require their vector difference $\boldsymbol{\omega}^{\leqslant} - \boldsymbol{\omega}^{>}$ to be normal to $\mathcal{H}$. From linear algebra, we know that the vector $\mathbf{a}$ is normal to $\mathcal{H}$. Let $\mathbf{a}^T\boldsymbol{\omega}^{\leqslant} - b = \delta$. We have $\mathbf{a}^T\boldsymbol{\omega}^{>} - b = -\delta$, and $\mathbf{a}^T(\boldsymbol{\omega}^{\leqslant} - \boldsymbol{\omega}^{>}) = 2\delta$. Since $\mathbf{a}$ and $\boldsymbol{\omega}^{\leqslant}$ are known, we can obtain $\boldsymbol{\omega}^{>} = \boldsymbol{\omega}^{\leqslant} - \frac{2\delta}{\|\mathbf{a}\|^2}\mathbf{a}$.

$\mathbf{Gen\_Discr}(\boldsymbol{\omega}^{\leqslant}, \boldsymbol{\omega}^{>}) \rightarrow \Delta_{\mathcal{H}}$. This algorithm accepts anchor points $\boldsymbol{\omega}^{\leqslant} = (\omega_1^{\leqslant}, \omega_2^{\leqslant}, \ldots, \omega_d^{\leqslant})^T$ and $\boldsymbol{\omega}^{>} = (\omega_1^{>}, \omega_2^{>}, \ldots, \omega_d^{>})^T$ corresponding to the hyperplane $\mathcal{H}$, and outputs the discriminator $\Delta_{\mathcal{H}}$. It first appends distance information to the anchor points, to obtain $\boldsymbol{\omega}_+^{\leqslant} = ((\boldsymbol{\omega}^{\leqslant})^T|(-0.5\|\boldsymbol{\omega}^{\leqslant}\|^2))^T$ and $\boldsymbol{\omega}_+^{>} = ((\boldsymbol{\omega}^{>})^T|(-0.5\|\boldsymbol{\omega}^{>}\|^2))^T$. Next, $\boldsymbol{\omega}_+^{\leqslant}$ and $\boldsymbol{\omega}_+^{>}$ are encrypted using M as $\langle\boldsymbol{\omega}^{\leqslant}\rangle = M^T\boldsymbol{\omega}_+^{\leqslant}$ and $\langle\boldsymbol{\omega}^{>}\rangle = M^T\boldsymbol{\omega}_+^{>}$. Finally, the algorithm selects a random positive value r, and generates the encrypted discriminator $\Delta_{\mathcal{H}} = r(\langle\boldsymbol{\omega}^{\leqslant}\rangle - \langle\boldsymbol{\omega}^{>}\rangle)$.

B. Halfspace Range Queries on Encrypted MBR Vertices

The cloud executes an encrypted halfspace range query on encrypted MBR vertices using following algorithms.

$\mathbf{Halfspace\_Qry}([\mathcal{V}], \Delta_{\mathcal{H}}) \rightarrow \mathcal{V}_{\mathcal{H}}^{\leqslant}$. This function accepts a set of encrypted MBR vertices $[\mathcal{V}]$ and a hyperplane discriminator $\Delta_{\mathcal{H}}$, and outputs the set $\mathcal{V}_{\mathcal{H}}^{\leqslant} = \mathcal{V} \cap \mathcal{H}^{\leqslant}$. It operates by calling the following function to test each point in $[\mathcal{V}]$.

$\mathbf{In\_Halfspace}([V], \Delta_{\mathcal{H}}) \rightarrow \{0, 1\}$ The function accepts an encrypted point $[V]$, a discriminator $\Delta_{\mathcal{H}}$, and outputs a bit indicating whether $V \in \mathcal{H}^{\leqslant}$ by computing $\Delta_{\mathcal{H}} \cdot [V]$. Since

$$\begin{aligned}
\Delta_{\mathcal{H}} \cdot [V] &= r\left(\langle\boldsymbol{\omega}^{\leqslant}\rangle - \langle\boldsymbol{\omega}^{>}\rangle\right) \cdot [V] \\
&= r\left((M^T\boldsymbol{\omega}_+^{\leqslant}) - (M^T\boldsymbol{\omega}_+^{>})\right)^T M^{-1}V_+ \\
&= r\left(\boldsymbol{\omega}_+^{\leqslant} - \boldsymbol{\omega}_+^{>}\right)^T V_+ \\
&= r\left(\|\boldsymbol{\omega}^{>} - V\| - \|\boldsymbol{\omega}^{\leqslant} - V\|\right),
\end{aligned}$$

$\Delta_{\mathcal{H}} \cdot [V] \geqslant 0$ iff V is in $\mathcal{H}^{\leqslant}$. The function outputs 1 iff V is in $\mathcal{H}^{\leqslant}$, and 0 otherwise.

C. Hyperrectangle Intersection

We show how to determine intersections between encrypted d-dimensional query and index hyperrectangles based on halfspace range queries. We require that each hyperrectangle face be orthogonal to a coordinate axis. That is, each face is hyperplane $\mathcal{H}_i = (x_1, \ldots, x_{i-1}, c_i, x_{i+1}, \ldots, x_d)$, where c_i is a constant, but the x_i are unconstrained. This constraint is needed since intersection tests using on halfspace range queries may not work on general polyhedra, as we will see.

An index hyperrectangle $\mathcal{R} \subset \mathbb{R}^d$ is now fully specified by its extremal vertices $V_\perp, V_\top \in \mathbb{R}^d$ defined as follows. If $V = (v_1, v_2, \ldots, v_d)$ represents a vertex of $\mathcal{R}$, then we define $V_\perp = (\min\{v_1\}, \ldots, \min\{v_d\})$, $V_\top = (\max\{v_1\}, \ldots, \max\{v_d\})$, where $\min$ and $\max$ are taken over all vertices V of $\mathcal{R}$.

A query hyperrectangle, however, is specified in terms of the halfspaces defined by its faces, as $\mathcal{Q} = \mathcal{H}_1^{\leqslant} \cap \mathcal{H}_2^{\leqslant} \cap \cdots \cap \mathcal{H}_{2d}^{\leqslant}$.

An index hyperectangle is encrypted as follows.

$\mathbf{Enc\_Index}(\mathcal{R}, M) \rightarrow [\mathcal{R}]$. Given an index hyperrectangle $\mathcal{R} = (V_\perp, V_\top)$, and key matrix M, this algorithm outputs the encryption of $\mathcal{R}$ as $[\mathcal{R}] = ([V_\perp], [V_\top])$, where $[V_\perp] = \mathbf{Enc\_Vertex}(V_\perp, M)$ and $[V_\top] = \mathbf{Enc\_Vertex}(V_\top, M)$.

$\mathbf{Enc\_Query}(\mathcal{Q}, M) \rightarrow \langle\mathcal{Q}\rangle$. This algorithm accepts the key matrix M and a query region $\mathcal{Q}$ specified as the intersection of halfspaces $\mathcal{H}_1^{\leqslant}, \mathcal{H}_2^{\leqslant}, \ldots, \mathcal{H}_{2d}^{\leqslant}$. For each $\mathcal{H}_i$, it first invokes $\mathbf{Gen\_Anchor}(\mathcal{H}_i)$ to get $\boldsymbol{\omega}_i^{\leqslant}$ and $\boldsymbol{\omega}_i^{>}$. It then obtains $\Delta_{\mathcal{H}_i}$ by invoking $\mathbf{Gen\_Discr}(\boldsymbol{\omega}_i^{\leqslant}, \boldsymbol{\omega}_i^{>})$. It returns the encrypted query region $\langle\mathcal{Q}\rangle = (\Delta_{\mathcal{H}_1}, \Delta_{\mathcal{H}_2}, \ldots, \Delta_{\mathcal{H}_{2d}})$.

$\mathbf{Xsect\_Index}([\mathcal{R}], \langle\mathcal{Q}\rangle) \rightarrow \{0, 1\}$. This function accepts an index hyperrectangle and a query hyperrectangle, both encrypted. It outputs a Boolean indicating whether the hyperrectangles intersect. If both $[V_\perp]$ and $[V_\top]$ are determined to lie outside $\mathcal{H}_i^{\leqslant}$ for some $\mathcal{H}_i$, the algorithm returns 0, and 1 otherwise (see Alg. 1).

Algorithm 1: Xsect_Index

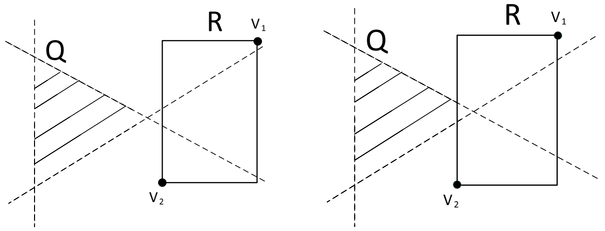
```
input  : $[\mathcal{R}] = ([V_\perp], [V_\top])$, $\langle\mathcal{Q}\rangle = (\Delta_{\mathcal{H}_1}, \ldots, \Delta_{\mathcal{H}_{2d}})$
output: $\{0, 1\}$
foreach $\Delta_{\mathcal{H}_i} \in \langle\mathcal{Q}\rangle$ do
    if not In_Halfspace($[V_\perp], \Delta_{\mathcal{H}_i}$) and
    not In_Halfspace($[V_\top], \Delta_{\mathcal{H}_i}$) then
        return 0
    end
end
return 1
```

D. Polyhedral Query Regions

Our scheme can handle arbitrary convex polyhedral query regions, but may introduce false positives. The two cases shown in Fig. 6 cannot be distinguished. Both cases return the same results if we run halfspace queries for the vertices of the index hyperrectangle using the $\mathcal{H}_i^{\leqslant}$ defining the triangular query region. However, our scheme is safe for convex polyhedral query regions, since it does not introduce false negatives according Theorem 1.

Theorem 1: $\mathbf{Xsect\_Index}([\mathcal{R}], \langle\mathcal{Q}\rangle)$ outputs 0 iff the convex polyhedral query region $\mathcal{Q}$ and the index range $\mathcal{R}$ do not intersect.

Proof. $\mathcal{R}$ is determined by $V_\perp = (\min\{v_1\}, \ldots, \min\{v_d\})$ and $V_\top = (\max\{v_1\}, \ldots, \max\{v_d\})$, its extremal vertices. If both $\mathbf{In\_Halfspace}([V_\perp], \Delta_{\mathcal{H}_i})$ and $\mathbf{In\_Halfspace}([V_\top], \Delta_{\mathcal{H}_i})$ output 0 for a halfspace query $\Delta_{\mathcal{H}_i} \in \langle\mathcal{Q}\rangle$, then neither $V_\perp, V_\top$ is in $\mathcal{H}^{\leqslant}$. Since $V_\perp$ and $V_\top$ have, respectively, the smallest and largest projection along axis i, none of $\mathcal{R}$'s 2^d vertices can be in $\mathcal{H}^{\leqslant}$. Clearly, $\mathcal{Q}$ and $\mathcal{R}$ do not intersect.



(a) No intersection (b) Intersection

Fig. 6: Halfspace queries using polyhedral region.

E. $\widehat{\mathrm{R}}$-tree Construction and Query

The $\widehat{\mathrm{R}}$-tree may be viewed as an R-tree whose MBRs are encrypted, but whose parent-children relationships are not. Its data points are encrypted separately. These encrypted data points, the encrypted $\widehat{\mathrm{R}}$-tree index, and the parent-children relationships are placed in the cloud. We test for overlaps between encrypted query ranges and encrypted $\widehat{\mathrm{R}}$-tree MBRs using the novel Encrypted Halfspace Query (EhQ) mechanism.

Our EhQ approach is secure and efficient, and may be used to search, under encryption, other complex data structures, such as BNL-trees and kD-trees. We have chosen to base our $\widehat{\mathrm{R}}$-tree index on the R-tree, since the R-tree family has lower information leakage than bucketization schemes, k-means, BNL-tree, kD-tree, and so on, as shown in [9].

$\widehat{\mathrm{R}}$-trees are constructed as in Alg. 2. Let $\mathcal{S}$ and $\widetilde{\mathcal{S}}$ denote the plaintext and encrypted versions of the data set. For a leaf's MBR $\mathcal{R}$, let $\mathcal{S}_{\mathcal{R}}$ denote the data points falling into $\mathcal{R}$, and $\widetilde{\mathcal{S}}_{\mathcal{R}}$ denote the ciphertexts of $\mathcal{S}_{\mathcal{R}}$. Let $\mathbf{T}$ and $\widehat{\mathbf{T}}$ denote an R-tree and the corresponding $\widehat{\mathrm{R}}$-tree, respectively. Let $\mathbf{PC}$ denote the set of parent-children relationships in $\mathbf{T}$.

The following function, elaborated in Alg. 3, finds all $\widehat{\mathrm{R}}$-tree leaves that intersect a given range query.

$\widehat{\mathbf{R}}$**-tree_Qry**$(\langle\mathcal{Q}\rangle, \widehat{\mathbf{T}}) \rightarrow \mathbf{L}$ This function takes as input an encrypted range query $\langle\mathcal{Q}\rangle$, an $\widehat{\mathrm{R}}$-tree $\widehat{\mathbf{T}}$. It outputs a set of encrypted leaves $\mathbf{L}$ whose MBRs intersect $\langle\mathcal{Q}\rangle$.

VI. Security Analysis

Our analysis shows that outsourcing schemes that leak the plaintext ordering of encrypted tuples cannot provide strong privacy guarantees. Such ordering information often permits the adversary to estimate the values of encrypted tuples quite accurately. We begin by analyzing the security of the scheme used for encrypting the index and query hyperrectangles in $\widehat{\mathrm{R}}$-trees. We then compare the privacy guarantees provided by our scheme with those provided by competing schemes, especially when the adversary has managed to discover partial information about the values of encrypted tuples.

A. Security of Encryption Schemes

We encrypt index and query hyperrectangles using ASPE, which has been proved secure under known-plaintext attacks in [14]. Our scheme retains the security properties of ASPE, since it extends ASPE, but does not alter the basic approach to encryption in [14]. Artificial dimensions and random asymmetric splitting still work in our scheme.

Algorithm 2: $\widehat{\mathrm{R}}$-tree_Construction

```
input  : $\mathbf{T}$, $M$
output: $\widehat{\mathbf{T}}$
$\widehat{\mathbf{T}} = \varnothing$
$\mathbf{PC} = \varnothing$ stack $= \varnothing$
node $= \mathbf{T}$.root
if node $\neq$ NULL then
    stack.Push(node)
end
else
    return $\varnothing$
end
while stack $\neq \varnothing$ do
    node $=$ stack.Pop()
    // node.$\mathcal{R}$ denotes node's MBR
    $[\mathcal{R}] = \mathbf{Enc\_Index}(node.\mathcal{R}, M)$
    if node has children then
        foreach child do
            Save the parent-child relationship to $\mathbf{PC}$
        end
    end
    if node is not a leaf then
        foreach child do
            stack.Push(child)
        end
        // Generate a node for $\widehat{\mathbf{T}}$
        onode $= \{[\mathcal{R}]\}$
    end
    else
        Encrypt data points in $\mathcal{S}_{\mathcal{R}}$ to obtain $\widetilde{\mathcal{S}}_{\mathcal{R}}$
        onode $= \{[\mathcal{R}], \widetilde{\mathcal{S}}_{\mathcal{R}}\}$
    end
    Add onode to $\widehat{\mathbf{T}}$
end
Add $\mathbf{PC}$ to $\widehat{\mathbf{T}}$
return $\widehat{\mathbf{T}}$
```

B. Comparisons With Competing Schemes

We will show how ordering information may be exploited by the adversary to infer plaintext values, using order statistics. As Table I shows, current schemes provide either efficiency or privacy protection, but not both. Thus, the bucketization scheme of [9] protects ordering information, but suffers from high query overhead. It also requires the data owner to manage bucket indices. In contrast, [4], [5] allow efficient queries, but reveal ordering information on encrypted tuples. The $\widehat{\mathrm{R}}$-tree is able to achieve very efficient queries, while hiding the ordering of data points within each leaf MBR.

For several reasons, we will not pursue a detailed comparison of our scheme with the bucketization scheme of [9]. First, [9] is not a true outsourcing scheme, since the index is kept at the data owner site, and not in the cloud. This

Algorithm 3: $\widehat{\mathrm{R}}$-tree_Qry

```
input : ⟨$\mathcal{Q}$⟩, $\widehat{\mathbf{T}}$
output: L
L = ∅
stack = ∅
node = $\widehat{\mathbf{T}}$.root
if Xsect_index(node.[$\mathcal{R}$], ⟨$\mathcal{Q}$⟩) then
    stack.Push(node)
end
else
    return ∅
end
while stack ≠ ∅ do
    node = stack.Pop()
    if node is a leaf then
        L = L ⋃ node
    end
    else
        foreach node's child do
            if Xsect_index(child.[$\mathcal{R}$], ⟨$\mathcal{Q}$⟩) then
                stack.Push(child)
            end
        end
    end
end
return L
```

Scheme	Query Overhead	Reveals Order?
Bucketization [9]	**High:** $O(N/C)$	No.
Order preserving [5]	Low: $\log N$	**Yes.**
Predicate encryption [4]	Low: $\log N$	**Yes.**

TABLE I: Encrypted database schemes. N is the number of tuples, C is the bucket size. (The $\log N$ overhead of [4], [5] can only be achieved for one-dimensional data.)

also requires all queries to be performed by the data owner, an onerous requirement. Finally, the index search takes time $O(N/C)$, which is linear in the database size, if we keep the bucket size constant. This overhead is excessive, compared with competing schemes.

We therefore compare the resilience of our scheme to that of the schemes in [4], [5]. This is an appropriate comparison, since the $\widehat{\mathrm{R}}$-tree achieves the same query time complexity as these schemes. We will show that our scheme has much better resilience, an advantage it holds against any scheme that does not hide tuple ordering.

C. Attack Model

Let $\mathcal{A}_O$ denote the adversary in the schemes revealing ordering information, and let $\mathcal{A}_{\widehat{R}}$ denote the adversary in our scheme. The index ranges of the $\widehat{\mathrm{R}}$-tree leaves are bounding boxes for clusters of encrypted data points. Higher-level nodes represent further aggregations of bounding boxes. $\mathcal{A}_{\widehat{R}}$ cannot see the index ranges in any of the $\widehat{\mathrm{R}}$-tree nodes, since they are encrypted with ASPE. However, $\mathcal{A}_{\widehat{R}}$ is able to learn the ordering of all leaf MBRs from enough query results. $\mathcal{A}_{\widehat{R}}$ can selectively choose halfspace discriminator $\Delta_{\mathcal{H}_i}$ from received queries to form new queries. But these queries can only help him obtain the ordering of leaf MBRs. The ordering of data points inside each leaf MBR is still secure.

$\mathcal{A}_{\widehat{R}}$'s goal is to infer the values of encrypted data points belonging to a leaf node λ_j of interest. $\mathcal{A}_{\widehat{R}}$ is able to learn the plaintext values of some encrypted data points. We assume $\mathcal{A}_{\widehat{R}}$ knows both the low-end and the high-end of the range in λ_j, and he also knows the distribution of points' values.

To compare our scheme with methods such as [4], [5] that reveal the ordering of encrypted points, we assume that $\mathcal{A}_O$ also tries to infer the values of encrypted data points in λ_j. $\mathcal{A}_O$ knows the relative ordering of all data points, the low-end and the high-end of the range in λ_j, and the distribution of points' values.

D. The Adversary's Optimal Estimator

The adversary's goal is to infer the values of encrypted tuples. To this end, he will use a statistical estimator, whose effectiveness must be measured in terms of the error it introduces. We will use the widely-used Mean Squared Estimation Error (MSEE) metric, also used in [9], which works as follows. For simplicity, we consider the one-dimension case.

One must often estimate the value of a random variable Y, itself inaccessible, in terms of a function $g(X)$ of an accessible random variable X. In our case, Y is a tuple's plaintext value. The adversary chooses an appropriate random variable X. The MSEE is defined as $\mathbb{E}[(Y - g(X))^2]$. The simplest choice for the adversary is $g(X) = c$, a constant. We find the value $c_{\min}$ of c that minimizes the MSEE as follows. Starting with

$$\min_c \mathbb{E}[(Y-c)^2] = \min_c \left\{\mathbb{E}[Y^2] - 2c \cdot \mathbb{E}(Y) + c^2\right\},$$

we differentiate with respect to c and set to 0, getting $c_{\min} = \mathbb{E}[Y]$. The minimum MSEE is now $\mathbb{E}[(Y-\mathbb{E}[Y])^2] = \mathit{Var}(Y)$. Therefore, the optimal estimator for Y is $\mathbb{E}[Y]$, which achieves the minimum MSEE value $\mathit{Var}(Y)$.

Hence, given an encrypted tuple $\widetilde{y}_i$ with plaintext value y_i drawn from a distribution modeled by the random variable Y, the best estimate the adversary can make for y_i is $\mathbb{E}[Y]$, achieving an MSEE of $\mathit{Var}(Y)$.

E. Mounting Attacks With and Without Ordering Information

Given a contiguous range $R = [y_s, y_e]$ containing $|R|$ encrypted tuples $(\widetilde{y}_1, \widetilde{y}_2, \ldots, \widetilde{y}_{|R|})$, both adversaries $\mathcal{A}_O$ and $\mathcal{A}_{\widehat{R}}$ try to infer plaintext values of encrypted tuples in R. We assume that both $\mathcal{A}_O$ and $\mathcal{A}_{\widehat{R}}$ know the plaintext values y_s, y_e of the two endpoints of the range R. Let the random variable Y follow the same distribution as the plaintext values of all encrypted tuples, having density function $f(y)$. Further, $\mathcal{A}_O$ knows both the distribution $f(y)$ of plaintexts y_i and the ordering of encrypted tuples $\widetilde{y}_i$. $\mathcal{A}_{\widehat{R}}$ knows the distribution $f(y)$, but not the ordering of encrypted tuples $\widetilde{y}_i$.

1) $\mathcal{A}_{\widehat{R}}$'s Attack (Ordering Unknown): Let the random variable Y_R follow the same distribution as the plaintext values of encrypted tuples in R. Using distribution $f(y)$, $\mathcal{A}_{\widehat{R}}$ finds the distribution $f_{Y_R}(y)$ for Y_R. We saw in Sec. VI-D that $\mathcal{A}_{\widehat{R}}$'s best estimator for any $\widetilde{y}_i \in R$ is $\mathbb{E}[Y_R]$.

2) $\mathcal{A}_O$'s Attack (Ordering Known): $\mathcal{A}_O$ can do much better, since he knows the ordering of the $\widetilde{y}_i$. $\mathcal{A}_O$ first finds $f_{Y_R}(y)$. Let the random variable $Y_{(k)|R}$ represent the plaintext value of the k^{th} smallest tuple in range R, having distribution $f_{Y_{(k)|R}}(y)$. $\mathcal{A}_O$ obtains $f_{Y_{(k)|R}}(y)$ using $f_{Y_R}(y)$ and Eqn. 1. Let $\widetilde{y}_{(k)}$ denote the k^{th} smallest tuple in R. As in Sec. VI-D, $\mathcal{A}_O$'s best estimator for $\widetilde{y}_{(k)}$'s plaintext $y_{(k)}$ is $\mathbb{E}[Y_{(k)|R}]$.

F. The Absolute Estimation Error Metric ε

Let $\mathcal{A}_{\widehat{R}}$ and $\mathcal{A}_O$ estimate the true plaintext value y_i for an encrypted tuple $\widetilde{y}_i \in R$ as $y_i^{\widehat{R}}$ and y_i^O, respectively. We define the *Absolute Estimation Error* (AEE) for $\mathcal{A}_{\widehat{R}}$ as $\varepsilon^{\widehat{R}}_{y_i} = |y_i - y_i^{\widehat{R}}|$ and for $\mathcal{A}_O$ as $\varepsilon^O_{y_i} = |y_i - y_i^O|$. If $\widetilde{y}_{(k)}$ is the k^{th}-smallest tuple in R, we define $\varepsilon^{\widehat{R}}_{(k)} = |y_{(k)} - \mathbb{E}[Y_R]|$ and $\varepsilon^O_{(k)} = |y_{(k)} - \mathbb{E}[Y_{(k)|R}]|$.

VII. Experiments

We demonstrate the dangers of revealing ordering information through experiments using real-world data sets. We use a "training" data set Z to estimate the distribution $f(y)$ of the plaintext attribute, and a target data set T representing the encrypted database, whose values we want to estimate. We will show that it is possible to estimate $f(y)$ from various other data sets that are quite accessible.

A. Experiment Overview

Let the target database T contain N encrypted tuples. We model the adversary's attack as follows.

1) Estimate the probability density of plaintext values from the training data. Call this density $\widehat{f}(y)$.
2) Sort target data. Select a range $R = [y_s, y_e]$ randomly, with $y_s < y_e$.
3) Restrict $\widehat{f}(y)$ to range R to obtain density $\widehat{f}_{Y_R}(y)$ and $\mathbb{E}[Y_R]$ for random variable Y_R.
4) Using the known ordering, apply order statistics to get density $\widehat{f}_{Y_{(k)|R}}(y)$ and $\mathbb{E}[Y_{(k)|R}]$ for each $Y_{(k)|R}$.
5) For each sorted tuple $y_{(k)} \in R$, obtain $\varepsilon^{\widehat{R}}_{(k)}$ and $\varepsilon^O_{(k)}$.

We will now show that this attack is realistic and practical, by demonstrating it on real-world data sets.

B. Estimating Data Distributions

Estimating a distribution $\widehat{f}(Y)$ from a data sample is an old problem [19]–[21]. Estimation is not our focus here; indeed, we will show that even crude estimation methods can give the adversary a big advantage. It can actually suffice to attack target data sets using distribution estimates derived from other similar data sets.

Data set	**#tuples**	**Salary range**	**Mean**	**StDev**
NBA 2004	404	\$325,000—\$27,696,430	3,878,766	4,051,396
NBA 2010	379	\$417,221—\$24,806,250	5,040,318	4,702,271

TABLE II: NBA salaries in 2004 and 2010

Say that we have obtained some other database Z that has the targeted attribute. For instance, sample salary data are published by the Census Bureau, and can serve as the basis for an attack on an encrypted database containing salaries. We treat Z's contents as a sample from Y's distribution, and sort it to get $z_{(1)}, z_{(2)}, \ldots z_{(N)}$. A crude estimate [19] for the cumulative distribution function $F(y)$ is the sample fraction $\leqslant y$, i.e.,

$$\widehat{F}(y) = \frac{1}{N}\sum_i \mathbf{1}\left(z_{(i)} \leqslant y\right). \tag{2}$$

We assume that the estimated distribution is uniform in each range $(z_{(i)}, z_{(i+1)}]$ where $z_{(i)} < z_{(i+1)}$, so that the estimated probability density in this range is

$$\widehat{f}(y) = \frac{\widehat{F}(z_{(i+1)}) - \widehat{F}(z_{(i)})}{z_{(i+1)} - z_{(i)}}, y \in (z_{(i)}, z_{(i+1)}] \tag{3}$$

We obtain $\widehat{f}(y)$ for $y \in [z_{(1)}, z_{(N)}]$ using Eqn. 3. For $y \notin [z_{(1)}, z_{(N)}]$, we set $\widehat{f}(y)$ to 0.

The adversary obtains the density function restricted to the range R as follows:

$$\widehat{f}_{Y_R}(y) = \frac{\widehat{f}(y)}{\int_{y_s}^{y_e} \widehat{f}(y)dy}, y \in R \tag{4}$$

Since the density in range R is now known, the adversary can apply Eqn. 1 to compute order statistics in this range.

C. Obtaining Absolute Estimation Error in Experiments

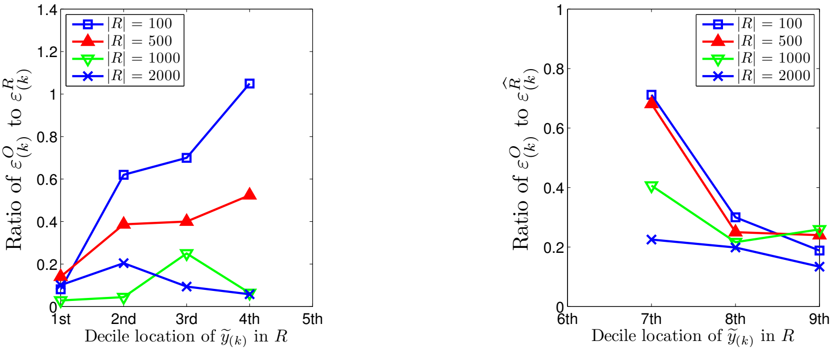
We proceed as follows to get the AEE for a tuple $y_{(k)} \in R$. For $\mathcal{A}_O$, we obtain $\mathbb{E}[Y_{(k)|R}]$ using order statistics, and get $\varepsilon^O_{(k)} = |y_{(k)} - \mathbb{E}[Y_{(k)|R}]|$. For $\mathcal{A}_{\widehat{R}}$, all we need is an estimate for $\mathbb{E}[Y_R]$. We can avoid the effects of errors made in estimating the distribution $\widehat{f}(y)$, by directly estimating $\mathbb{E}[Y_R]$ using the sample mean $\hat{\mu} = \left(\sum_{i=1}^{|R|} y_i\right)/|R|$. We can now compute $\varepsilon^{\widehat{R}}_{(k)} = |y_{(k)} - \hat{\mu}|$.

D. Tests Using Real-World Data Sets

Our first real-world data set is the Michigan income data set from the Census Bureau [22]. It contains 77,681 values, with a range of (\$-10,000, \$695,000), a mean of \$27,050, and standard deviation \$36,616. We randomly select half the tuples as our training set, and the rest as our target set.

Our second data set, shown in Table II, is the NBA player salaries in 2004 and 2010 [23]. We estimate the distribution from the 2004 salaries, and try to infer values in the 2010 salary data, which serves as our target.

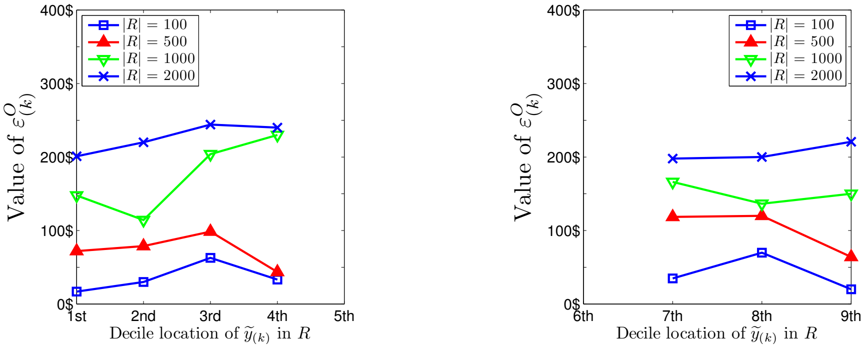
For the Michigan data set, we randomly select ranges R where $|R|$, the number of tuples in R, is 100, 500, 1000, and 2000. For each $\widetilde{y}_{(k)}$, we obtain the ratio of $\varepsilon^O_{(k)}$ to $\varepsilon^{\widehat{R}}_{(k)}$. We repeat the experiment 100 times for each $|R|$ value, and obtain the average ratio. We also find the average value of $\varepsilon^O_{(k)}$ for this data set.



(a) 1st to 4th decile

(b) 7th to 9th decile

Fig. 7: Average ratio of $\varepsilon^{O}_{(k)}$ to $\varepsilon^{\widehat{R}}_{(k)}$ in Michigan data set.



(a) 1st to 4th decile

(b) 7th to 9th decile

Fig. 8: Average Value of $\varepsilon^{O}_{(k)}$ in Michigan data set.

Fig. 7 shows the ratio $\varepsilon^{O}_{(k)}/\varepsilon^{\widehat{R}}_{(k)}$ for the tuple at each decile in the range R. The $\widehat{\text{R}}$-tree has much higher AEE than schemes that reveal the ordering of encrypted tuples, so the $\widehat{\text{R}}$-tree provides superior privacy protections. Fig. 7a shows the AEE ratios for tuples between the first and fourth deciles, and Fig. 7b shows the AEE ratio for tuples between the seventh and ninth deciles. It is clear that even with a very simple estimated distribution, $\mathcal{A}_O$'s estimates are much better than $\mathcal{A}_{\widehat{R}}$'s.

We do not show AEE ratios for tuples at the fifth and sixth deciles, since ordering information does not improve estimation accuracy in this range. $\mathcal{A}_{\widehat{R}}$'s estimation error $\varepsilon^{\widehat{R}}_{(k)} = |y_{(k)} - \hat{\mu}|$ is small when $y_{(k)}$ is close to $\hat{\mu}$, as is the case for tuples in this decile range. Since $\varepsilon^{O}_{(k)}$ is derived from coarse distribution estimates, it is not surprising that $\varepsilon^{\widehat{R}}_{(k)}$ is better than $\varepsilon^{O}_{(k)}$ for tuples in the middle deciles.

Fig. 8 shows that the average AEE value for $\mathcal{A}_O$'s is only hundreds of dollars. Given that mean income is $27,050, this is a serious violation of privacy in an encrypted database. Revealing the ordering can be disastrous. From Fig. 7 and Fig. 8, we can see that the average AEE value for $\mathcal{A}_{\widehat{R}}$'s is much higher than that for $\mathcal{A}_O$'s, especially for tuples with small and large orders. It is clear that hiding ordering information in $\widehat{\text{R}}$-tree enhances data privacy.

We ran similar experiments with the NBA income data for ranges of width 20, 50, 100, and 200. The training data were salaries in 2004, and the target data were the 2010 salaries. Fig. 9 shows that our 2010 salary estimates are quite good, given ordering information. Clearly, it suffices to use old public data to analyze new, and confidential data. It also shows that our estimation method works well on smaller data sets.

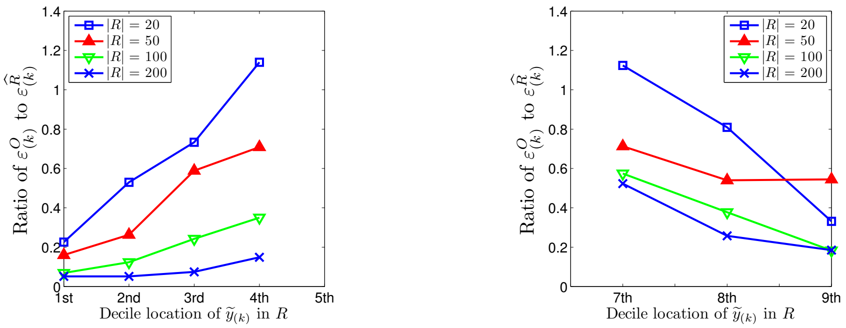
VIII. Performance Analysis

Our approach separates the $\widehat{\text{R}}$-tree index from the data, and encrypts each separately. Since the data is encrypted with a method of the data owner's choosing, data space requirements depend on the encryption scheme used.

We focus on analyzing the performance of the $\widehat{\text{R}}$-tree index, which is encrypted with ASPE. Even if the index vertices are inherently d-dimensional, security considerations in ASPE require that we extend their dimensionality to $d' \geqslant 80$ using extra artificial dimensions. Each index vertex is therefore presented as a d'-dimensional vector to ASPE.

However, these extra dimensions do not affect parent-children relationships, which continue to exist in $\mathbb{R}^d$. Thus the R-tree underlying the $\widehat{\text{R}}$-tree remains d-dimensional, and $\widehat{\text{R}}$-tree performance does not degrade due to high dimensionality.

Following [24], we normalize query ranges to $[0, 1]$, and proceed to define the *density* $D(N, s)$ of a set of N d-



(a) 1st to 4th decile

(b) 7th to 9th decile

Fig. 9: Average Ratio of $\varepsilon^{O}_{(k)}$ to $\varepsilon^{\widehat{R}}_{(k)}$ in NBA data set.

dimensional data objects with average size s as

$$D(N, s) = \sum_{i=1}^{N} s = Ns. \tag{5}$$

Let h denote the $\widehat{\text{R}}$-tree height and f denote the fanout (capacity) of each node. Let data points be at level 0, leaves at level 1, and the root at level h. Let the number, average size, and density of objects at level j be N_j, s_j, and D_j, respectively. As shown in [24], the density D_{j+1} is

$$D_{j+1} = N_{j+1}s_{j+1} = \left(1 + \frac{\sqrt[d]{D_j} - 1}{\sqrt[d]{f}}\right)^d \tag{6}$$

While the objects considered in [24] may be points or have extent, we consider only points. Since points have size 0, we have $D_0 = 0$, and

$$D_1 = (1 - f^{-\frac{1}{d}})^d \tag{7}$$

A. Storage Costs

Let an $\widehat{\text{R}}$-tree have N_j nodes at level j, each of capacity f. $N_{\widehat{\mathbf{T}}}$, the total number of nodes from level 1 to level h, is

$$N_{\widehat{\mathbf{T}}} = \sum_{j=1}^{h} N_j = \frac{f^h - 1}{f - 1},$$

where $h = 1 + \lceil \log_f N - 1 \rceil$. Since each node contains two d'-dimensional extremal vertices, the total storage needed is $4d'N_{\widehat{\mathbf{T}}}e$, where e denotes the size of each element in a vertex. The factor 4 arises due to random splitting [14].

B. Query Computation Costs

We first turn to the algorithm **Xsect_Index**. This function performs at most d EhQs for d-dimensional data. Let $C_{\mathbf{Xsect}}$ be the cost of **Xsect_Index**, and C_1 be the cost of an EhQ query on an index vertex. C_1 incurs the cost of two vector inner-products in $\mathbb{R}^{d'}$, each of which requires d' multiplications and $d' - 1$ sums. We have $C_{\mathbf{Xsect}} = O(dC_1) = O(dd')$.

We now consider the costs of $\widehat{\text{R}}$-tree range queries. An $\widehat{\text{R}}$-tree's structure is identical to that of its underlying R-tree. The $\widehat{\text{R}}$-tree uses the R-tree query algorithm, except that it uses **Xsect_Index** to perform intersection tests on encrypted data and queries do not descend to the level of data points. If a query $\mathcal{Q}$ on an R-tree requires $X_{\mathcal{Q}}$ intersection tests from level 1 to level h, then the cost of running the same query on the corresponding $\widehat{\text{R}}$-tree is $O(dd'X_{\mathcal{Q}})$.

C. Counting False Positives

If $\mathcal{Q}$ has normalized range length q_i on the ith dimension, let the false positive rate $\mathbf{FP}(N, f, \mathcal{Q})$ be defined as the fraction of returned query results that are spurious. As in [24], DA_j, the number of nodes covered at level j by query $\mathcal{Q}$ is

$$DA_j = \frac{N}{f^j} \prod_{i=1}^{d} \left(\left(D_j \frac{f^j}{N} \right)^{\frac{1}{d}} + q_i \right). \tag{8}$$

DA_0 is the actual count of data points matching the query, and DA_1 is the number of level-1 nodes matching the query. Since each level-1 node contains f level-0 points, we have

$$\mathbf{FP}(N, f, \mathcal{Q}) = 1 - \frac{DA_0}{fDA_1} = 1 - \frac{\prod_{i=1}^{d} q_i}{\prod_{i=1}^{d} \left(\left(D_1 \frac{f}{N} \right)^{\frac{1}{d}} + q_i \right)}$$

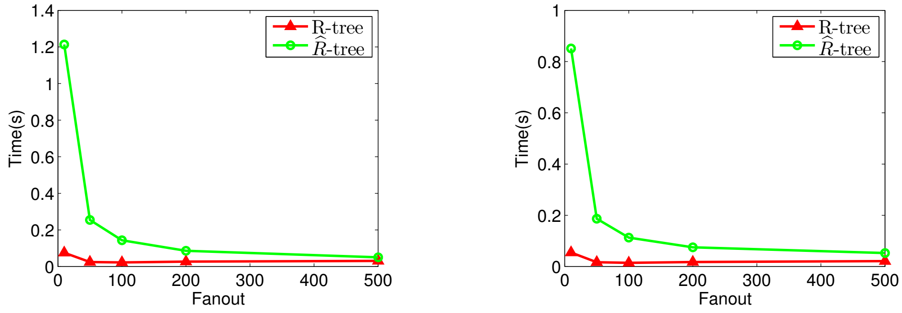
For uniform queries with $\mathbb{E}(q_i) = \frac{1}{\kappa}$ for $1 \leqslant i \leqslant d$, we can estimate the spurious fraction in the returned results as

$$\begin{aligned} \widehat{\mathbf{FP}}(N, f, \mathcal{Q}) &= 1 - \frac{\prod_{i=1}^{d} \mathbb{E}(q_i)}{\prod_{i=1}^{d} \left(\left(1 - \frac{1}{f^{\frac{1}{d}}} \right) \left(\frac{f}{N} \right)^{\frac{1}{d}} + \mathbb{E}(q_i) \right)} \\ &= 1 - \frac{N}{\left(\kappa f^{\frac{1}{d}} + N^{\frac{1}{d}} - \kappa \right)^d} \end{aligned} \tag{9}$$

IX. Our $\widehat{\text{R}}$-tree Implementation

We conducted extensive empirical analysis of an actual $\widehat{\text{R}}$-tree implementation built upon libspatialindex [25].

We set d' to 100 in ASPE to ensure security. We measured the $\widehat{\text{R}}$-tree false positive rate and query time in our experiments, and compared our measures with theoretical predictions. We also compared the $\widehat{\text{R}}$-tree algorithms to those in predicate-encryption based schemes such as [4], [12]. At



(a) SF

(b) NA

Fig. 10: Average query running time of R-tree and $\widehat{\text{R}}$-tree

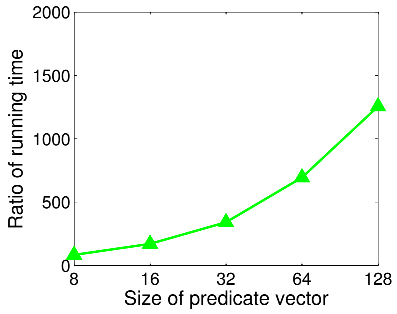


Fig. 11: Predicate encryption VS **Enc_Vertex**.

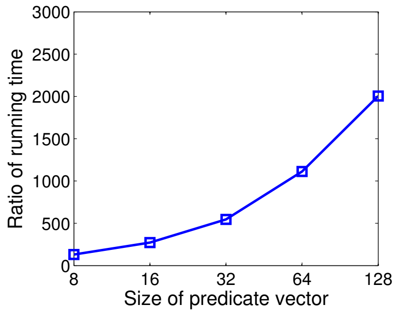


Fig. 12: Predicate token generation VS **Enc_Query**.

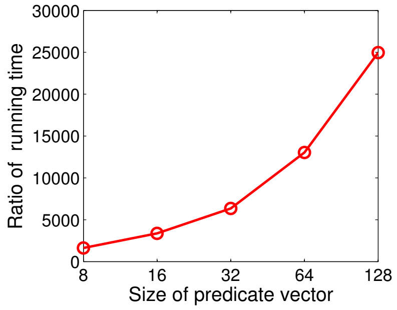


Fig. 13: Predicate decryption VS **Xsect_Index**.

Dataset	Number of points	Dimension
NA	175813	2
SF	174956	2

TABLE III: Data sets

last, we measured the AEE for $\widehat{\text{R}}$-tree's leaf MBRs. Our experiments ran on a Linux server with four 2GHz cpus and 8G memory.

Since ASPE must invert random matrices, it creates matrices with rational entries. Low floating point precision affects query results. Simply using double precision in C++ caused a lot of false positives and false negatives to appear in the results. We therefore used the GMP Library [26] to support high precision matrix operations.

A. *Data Sets Used*

We used real-world spatial data sets in our experiments, including road network data for North America (NA) and San Francisco (SF) [3], [27]. Table. III gives the details. We normalized the data range along each dimension to $[0, 1]$.

B. *Query Time*

The $\widehat{\text{R}}$-tree provides a high level of security, which traditional spatial indexing methods, such as the R-tree, do not. We would therefore expect to see that $\widehat{\text{R}}$-trees have higher query time than completely insecure methods.

Fig. 10 shows the impact of encryption on average query time. When the fanout is small, the average query time of $\widehat{\text{R}}$-tree is significantly higher than that of the R-tree, but as fanout increases, the average $\widehat{\text{R}}$-tree query time drops significantly, and is only about 0.1s longer than that of R-tree. This is a very good result, since security concerns preclude small fanouts, and cloud service providers generally have adequate computing capacity.

1) Comparisons with Predicate-Encryption-based Schemes: To compare the query times of the $\widehat{\text{R}}$-tree and predicate-encryption based schemes (PRE) [4], we implemented the encryption, decryption and token generation algorithms in PRE using PBC library [28]. These accomplish the same functions as **Enc_Vertex**, **Xsect_Index** and **Enc_Query** algorithms in the $\widehat{\text{R}}$-tree.

Figs. 11 to 13 show the ratio of running times for PRE's algorithms to those of the $\widehat{\text{R}}$-tree. Clearly, $\widehat{\text{R}}$-tree performance is vastly superior to that of PRE. The X-axis in these figures show the size of the predicate vector used in PRE. If the predicate vector size is u, PRE supports no more than 2^u distinct values in the range. PRE's encryption and token generation algorithms are 1200 and 2000 time slower than the corresponding $\widehat{\text{R}}$-tree algorithms. Decryption in PRE is as much as 25000 times slower than the $\widehat{\text{R}}$-tree algorithm to test for MBR intersections.

Fundamentally, our intersection test algorithm only requires a handful of vector inner-product operations, while the decryption algorithm in predicate-encryption requires expensive

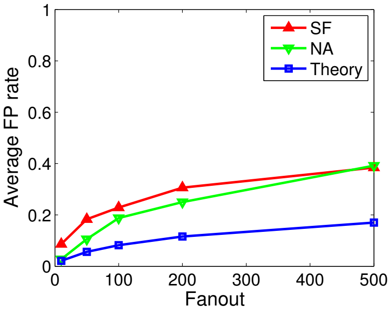


Fig. 14: False positive rate.

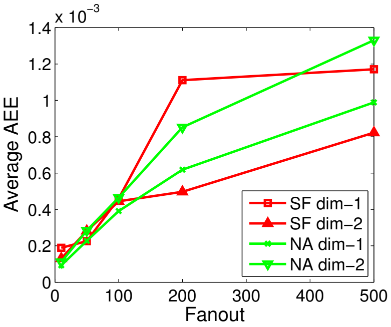


Fig. 15: Average AEE for $\widehat{\text{R}}$-tree.

bilinear-paring operations.

C. False Positive Rate

False positives increase privacy, so a significant false positive rate is not a huge problem in a secure outsourcing scheme such as the $\widehat{\text{R}}$-tree. As we have noted, data owners can tune the $\widehat{\text{R}}$-tree for different tradeoffs between security and false positives. Fig. 14 shows the average false positive rate for 1000 queries over the data sets NA and SF, and the theoretical false positive rate of uniformly distributed data for different fanouts. The false positive rate is usually less than 50%.

D. AEE for $\widehat{\text{R}}$-tree

Fig. 15 shows the average AEE of leaf MBRs for the NA and SF data set. For each leaf MBR, we obtain $\mathcal{A}_{\widehat{R}}$'s AEE separately for each of the two dimensions (our data set is 2-dimensional). Next, we average this AEE over all leaf MBRs, separately along each dimension. Figs. 10, 14 and 15 show that the false positive rate increases with fanout, but so does AEE, which increases security. For higher fanouts, $\widehat{\text{R}}$-tree query times get closer to that of ordinary R-trees.

X. Conclusion

In this paper, we present $\widehat{\text{R}}$-tree, a hierarchical encrypted index that can achieve secure and efficient range queries on encrypted data. $\widehat{\text{R}}$-tree hides ordering inside leaf MBRs to protect data privacy. Our theory and empirical analysis show revealing ordering is dangerous for outsourced data, and $\widehat{\text{R}}$-tree has much better resilience than schemes without ordering information protection. We also develop a system implementing $\widehat{\text{R}}$-tree, and find its performance is quite good.

XI. Acknowledgement

This work was supported in part by contract number N00014-07-C-0311 with the Office of Naval Research.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[2] J. Katz and Y. Lindell, *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[3] M. L. Yiu, G. Ghinita, C. S. Jensen, and P. Kalnis, "Outsourcing search services on private spatial data," in *ICDE*, 2009, pp. 1140–1143.

[4] Y. Lu, "Privacy-preserving logarithmic-time search on encrypted data in cloud," in *NDSS*, 2012.

[5] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *EUROCRYPT*, 2009, pp. 224–241.

[6] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *EUROCRYPT*, 2008, pp. 146–162.

[7] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," *IACR Cryptology ePrint Archive*, vol. 2006, p. 287, 2006.

[8] E. Shi, J. Bethencourt, H. T.-H. Chan, D. X. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *IEEE Symposium on Security and Privacy*, 2007, pp. 350–364.

[9] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," *The VLDB Journal*, vol. 21, pp. 333–358, 2012.

[10] H. A. David and H. N. Nagaraja, *Order Statistics, Third Edition*. New York: Wiley, 2003.

[11] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *TCC*, 2009, pp. 457–473.

[12] M. Li, S. Yu, N. Cao, and W. Lou, "Authorized private keyword search over encrypted data in cloud computing," in *ICDCS*. IEEE Computer Society, 2011, pp. 383–392.

[13] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," in *VLDB*, 2004, pp. 720–731.

[14] W. K. Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *SIGMOD Conference*, 2009, pp. 139–152.

[15] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *CRYPTO*, 2011, pp. 578–595.

[16] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*. ACM, 1984, pp. 47–57.

[17] A. C.-C. Yao and F. F. Yao, "A general approach to d-dimensional geometric queries (extended abstract)," in *STOC*, 1985, pp. 163–168.

[18] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[19] E. Parzen, "On estimation of a probability density function and mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[20] M. Rudemo, "Empirical choice of histograms and kernel density estimators," *Scandinavian Journal of Statistics*, vol. 9, no. 2, pp. pp. 65–78.

[21] A. B. Tsybakov, *Introduction to Nonparametric Estimation*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[22] "United states census bureau product catalog." [Online]. Available: http://www.census.gov/mp/www/cat

[23] "Usatoday salaries databases." [Online]. Available: http://content.usatoday.com/sportsdata/basketball/nba/salaries/team

[24] Y. Theodoridis and T. K. Sellis, "A model for the prediction of r-tree performance," in *PODS*, 1996, pp. 161–171.

[25] "libspatialindex." [Online]. Available: http://libspatialindex.github.com

[26] "Gmp: The gnu multiple precision arithmetic library." [Online]. Available: http://gmplib.org

[27] T. Brinkhoff, "A framework for generating network-based moving objects," *Geoinformatica*, vol. 6, pp. 153–180, June 2002.

[28] B. Lynn, "Pbc: The pairing-based cryptography library." [Online]. Available: http://crypto.stanford.edu/pbc