UDC 004.75 DOI: 10.30748/nitps.2018.31.16

Salman Rasheed Owaid

AL-Maaref University College, Iraq

SECURITY OF ENCRYPTED CLOUD DATABASE

Existing cipher method provides either security or productivity, but not both of these parameters. Most of schemes even show the order of encrypted sequence, which allow the detractors to accurately evaluate clear. This work represents $\hat{R}$ *as the tree, hierarchical encrypted index, which can be reliably put into a cloud and effectively falsified. It is based on mechanism, developed for encrypted queries, using Asymmetric scalar product preserving encryption (ASPE).*

Keywords: *query encryption, database encryption, cloud computing, hierarchical index.*

Introduction

This article represents $\hat{R}$-tree, hierarchical encrypted index, which can be reliably put into a cloud and effectively falsified. It is based on mechanism, developed for encrypted queries of half-space range in R^d, using Asymmetric scalar product preserving encryption (ASPE). Data owners can configure parameters of $\hat{R}$-tree to achieve required security and productivity. Also we represent experiments in performance assessment of $\hat{R}$-tree. Our results show, that queries of $\hat{R}$-trees are performed in encrypted databases and show much less information than competing methods.

Problem state

Term «cloud computing» is related to a wide range of services outsourced for storage and computing [1]. This model becomes more and more popular, because users have practically unlimited resources, but more significantly, they are exempt from the burden of managing these resources. That is why large database outsourcing became a well-studied topic.

However, this model has costs. Outsource data should be encrypted to preserve confidentiality and integrity, but encryption aggravates query execution. Standard encryption schemes, such as encryption units, do not directly support collation comparison, search and other manipulations, required for processing queries without loss of confidentiality. Therefore, new encryption schemes were proposed [2–5] to facilitate queries on encrypted data.

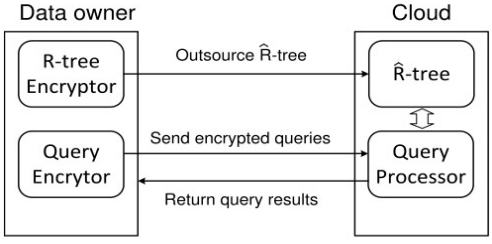


Fig. 1. Scheme of the search model based on $\hat{R}$-tree

Security and efficiency are important considerations when developing such encryption schemes. Some schemes [2; 4] achieve productivity by detecting the relative order of the encrypted data points, but the detractor can use information for data reorganization, using order statistics [6].

Query schemes based on predicate-based encryption (PRE) [4; 7–9] provide reliable protection for encryption, but with high computational costs. The costs in these schemes significantly increase with the range of queries or required accuracy.

Some leakage of data on the order is probably unavoidable, but the task is to minimize such leakage. For example, bouquet schemes [3] provide an exchange between ordering of the data leakage about the order and productivity.

1. Research objective

Innovative method for performing encrypted queries of the half-space range in R^d by points, encrypted with ASPE. This mechanism can provide multifaceted requests for encrypted data. Using this mechanism, we represent $\hat{R}$-tree, indexing scheme for encrypted and outsourced databases. $\hat{R}$-tree uses ASPE to encrypt query ranges. The data can be encrypted in any other way. $\hat{R}$-tree is a hierarchical scheme of , but unlike the current bouquet schemes, encrypted indexes of $\hat{R}$-tree are stored and are fully requested in the cloud, but not on the site of database owner. $\hat{R}$-trees gives us an opportunity to transfer data to data management more effective.

2. ASPE Asymmetric encryption

In [10] Asymmetric scalar product preserving encryption (ASPE) for kNN queries execution in encrypted data points in R^d was proposed. Encryption uses $(d+1)\times(d+1)$ as a secret key M invertible matrix. Data and queries are encrypted in different ways, which is reflected in our notations.

$\text{Point\_Enc}(P,M) \to \langle P \rangle$. This function accepts data point $P \in R^d$ and $(d+1)\times(d+1)$ of key matrix M and outputs the encrypted text $\langle P \rangle$ from P. Firstly it creates a point $P_+ \in R^{d+1}$, when $P_+ = (P^T \mid (-0.5 \| P \|^2))^T$ is Euclidian norm P. Encryption P is equal to $P = M^T P_+$.

$\text{Query\_Enc}\left(Q, M^{-1}\right) \to [Q]$. This function takes a request point $Q \in R^d$ and M^{-1}, inverse of the key matrix M. It produces $[Q]$, encrypted text Q. Firstly it creates point $Q_+ \in R^{d+1}$, that $P_+ = (P^T \mid (-0.5 \| P \|^2))^T$, where r is a random positive number. Encrypted point $[Q] = M^{-1} Q_+$.

$\text{Dist\_Comp}(\langle P \rangle, \langle P' \rangle, [Q]) \to \{0,1\}$. This function takes two encrypted data points $\langle P \rangle, \langle P' \rangle$, encrypted query point $[Q]$ returns 1, if P is closer to Q, than P'. It outputs a logic value $(\langle P \rangle - \langle P' \rangle) \cdot [Q] > 0$. Now,

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

where $\| P - Q \|$ is Euclidean distance between P and Q. This expression is positive, if P is closer to Q, than P'. Request kNN identifies k of the closest points by comparing the distance from the query point Q to each database point P.

2.1. Queries of the half-space range

The half-space Range Queries of (hRQ) are a fundamental problem in computational geometry, as any search form for algebraical range can be transformed into it. If $\mathbf{a} \in \mathbf{R}^d$, $\mathbf{a} \neq 0$ and $b \in \mathbf{R}$, then hyperplane H is determined by the set $\mathbf{x} \in R^d$ that $\mathbf{a}^T \mathbf{x} = b$. H splits $\mathbf{R}^d$ for inner half-space $H^{\leq}$, corresponding to $\mathbf{a}^T \mathbf{x} \leq b$ and external half-space $H^{>}$, corresponding to $\mathbf{a}^T \mathbf{x} > b$. Each $S \in \mathbf{R}^d$ is splitted by H into two disjoint subsets $S_H^{\leq} = S \cap H^{\leq}$ and $S_H^{>} = S \cap H^{>}$.

Considering the set of points $S = \{P_1, P_2, ..., P_n\}$ and hyperplane H in $\mathbf{R}^d$, a query of the half-range requests $S_H^{\leq}$.

2.2. Order Statistics

Order statistics is an important tool in nonparametric statistics [6]. Let $X_1, X_2, ..., X_n$ be random variables with density and distribution functions $f(x)$ and $F(x)$ correspondently. Let X_i sorting, in order to get $X_{(1)} \leq X_{(2)} \leq ... \leq X_{(n)}$. Now X_k is the statistics of k order. It can be shown that the density function X_k is provided by the formula:

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

2.3. Review

Now we will present our system and security model and provide the overview for R-tree. Our approach, in contrast to [10], uses index to speed up the queries. In addition, we separate the encryption of data points from query data and index.

2.3.1. System model

Our model recognizes two objects: the data owner and the provider of cloud services (fig. 2). The data owner places the encrypted data and the corresponding index of R-tree in cloud, which provides an infrastructure for computing and storage. The data owner creates and sends encrypted requests for ranges of interest to the cloud. The cloud performs encrypted queries in the index of R - tree and returns the query results to the data owner.

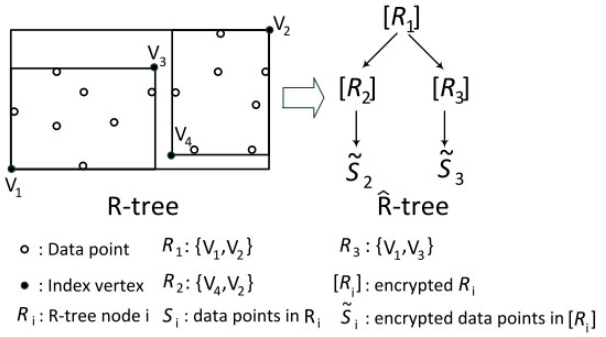


Fig 2. R-tree and $\hat{R}$-tree

The data owner creates $\hat{R}$-tree, first creating a regular R-tree for a given set of points $S \in \mathbf{R}^d$. Ranges MBR are encrypted using ASPE to get $\hat{R}$-tree. Parent-child relationship in $\hat{R}$-tree is not encrypted. Although MBR ranges of R-tree are encrypted using ASPE for range requests support, data points in S can be independently encrypted with other encoding schemes, such as block encryption. d-dimensional range R can be defined by its two extremal vertices. R-tree in fig. 2 contains nodes with MBR $R_1 = (V_1, V_2)$, $R_2 = (V_1, V_2)$ and $R_3 = (V_1, V_2)$. Data set, containing in R_2 and R_3, represents S_2 and S_3 correspondently.

MBR ranges in $\hat{R}$-trees are encrypted by applying ASPE to each extremal vertex, used to determine the range. As shown in fig. 2, correspondent $\hat{R}$-tree con-

tains three nodes $[R_1]=([V_1],[V_2])$, $[R_2]=([V_4],[V_2])$, $[R_3]=([V_1],[V_3])$.

The data points inside each MBR leaf are encrypted using regular encryption scheme. Encrypted versions of S_2 and S_3 are $\tilde{S}_2$ and $\tilde{S}_3$.

The data owner creates encrypted range requests and sends them to the cloud. The cloud searches $\hat{R}$-tree, performing intersection tests in stages, as in regular R-tree, and moves to child nodes only in case, when the bounding box of the node crosses the range of queries. The cloud thereby receives all the layers that cross the range of queries and returns the encrypted data points on these leafs to the data owner.

The cloud cannot query namely data points, because they are separately encrypted.

$\hat{R}$-tree can enter false positives in the query results, but protects order information within each leafs MBR is a reasonable compromise. Exact query schemes [4; 8], which return exactly the set of encrypted tuples in the query range, can leak information over time. Given the sufficient results of the query range, the enemy can restore the ordering of tuples from the joins and intersections of these result clusters.

2.3.2. Security Model

We accept an "honest, but curious" model for our detractor, the cloud server. Its purpose is to study open texts for encrypted data. It can contain some knowledge about the outsourcing dataset and try to use this knowledge to get the values of points in the dataset. Otherwise, it scrupulously monitors the protocol defined by the data owner, and returns the correct query results.

ASPE, which we use to encrypt ranges of indexes and queries, is protected from known attacks with plaintext [10]. However, the detractor can create more difficult attacks. For example, it can get some information about ordering encrypted data values when processing requests. The detractor may know the distribution of the plaintext values of all data points and the values of some data points. He will try to evaluate the values of other data points using this information.

Let's start with assumption that the detractor knows the order of the encrypted data points. Some encryption schemes, such as [2; 4], clearly show this order. In other cases, it may be possible to derive this ordering from time to time from queries. It is also often possible to obtain data value distributions from either public sources or by examining other available and similar data sets.

Using such knowledge of distributions and ordering, the opponent can use the order statistics methods to estimate the values of plaintext for encrypted tuples. For one-dimensional data, let's say that the enemy is studying the values of plain text m of $y_{i1} < y_{i2} < ... < y_{im}$ data points. It uses these points as end points to obtain ranges $m-1$ $[y_{i1}, y_{i2}],...,[y_{im-1}, y_{im}]$. It knows the order of encrypted tuples in each range and can now get better estimates of the plaintext values of the encrypted tuples in each range using order statistics. For multi-dimensional data, the enemy can perform the same attack to better estimate the values of the encrypted tuples for each dimension.

$\hat{R}$-trees do not disclose the full order of data points, but contain information on the leakage of information on the ordering of leaf MBR. We will study the effectiveness of the attacks described above on $\hat{R}$-trees.

2.3.3 Request of half-space range for encrypted data

Requests in [10] ask, which encrypted data points in $\mathbf{R}^d$ is the closest to the specified encrypted request point. Their method converts each data point into a $\mathbf{R}^d$ to the point in $\mathbf{R}^{d+1}$, additional dimension, encoding the distance of a point from the origin. However, the request points are not required to transmit such distance information. Our approach to encrypted half-space queries (EhQ) is dual to this method and must check which of the two query points is closer to the vertex in MBR of $\hat{R}$-tree. Therefore, the query points are embedded with the distance information in our scheme, whereas the points corresponding to the MBR vertices are not present.

We construct queries on the half-space, as in fig. 3. For a hyperplane H and the corresponding half-spaces $H^{\leq}$ and $H^{>}$ we choose reference points $\omega^{\leq} \in H^{\leq}$ and $\omega^{>} \in H^{>}$, equidistant from H, such, that the segment $(\omega^{>}, \omega^{\leq})$ is orthogonal to H. Each point on H is now equidistant from $\omega^{\leq}$ and $\omega^{>}$, but points in $H^{\leq}$ are closer to $\omega^{\leq}$, and points in $H^{>}$ are closer to $\omega^{>}$. We can check if the given point is located in V, $H^{\leq}$ or in $H^{>}$, proving, if V is closer to $\omega^{\leq}$ or to $\omega^{>}$, as in ASPE.

2.3.4 Index search as the intersection of hyper-intersections

Search of $\hat{R}$-trees require to know, in order to determine, if hyperrectangle intersects d-dimensional query hyperrectangle index in the node of $\hat{R}$-tree. We make the usual assumptions that the coordinate axis is orthogonal and that each face of the hyperrectangle is orthogonal to some axis.

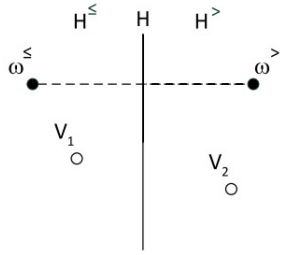


Fig. 3. Query of a half-space range

Our method is based on the observation that d-dimensional hyperrectangle of Q query can be defined as a space, enclosed by hyperplanes $H_1, H_2, ..., H_{2d}$, determined by its 2d borders. We accept the agreement that the scope of the request is defined in $H_i^{\leq}$ for each i. That is, the hyperplanes are so precise that the points of interest $\mathbf{x}$ satisfy the condition $\mathbf{a}_i^T \mathbf{x} \leq b_i$. In these conditions we will have $Q = H_1^{\leq} \cap H_2^{\leq} \cap \ldots \cap H_{2d}^{\leq}$. Fig. 4 shows a two-dimensional query box, specified by four half-space range queries, or eight anchor points. Half-spaces $H_i^{\leq}$ and $H_i^{>}$ are defined by two reference points $\omega_i^{\leq}$ and $\omega_i^{>}$. Choose $\omega_i^{\leq}$ randomly in $H_i^{\leq} - H_i$. $\omega_i^{>}$ will be its reflection in the hyperplane H_i. We indicate each index of a hyper-rectangle $R \subseteq \mathbf{R}^d$ in the node of $\hat{R}$-tree with its two peaks, as shown in fig. 2.

On the fig. 4 three cases of two-dimensional rectangular intersections are shown. Clearly, we cannot test the intersection of rectangles simply by verifying whether one vertex is included in the other. Instead, we need to check if the vertices in the corresponding half-spaces are defined by the faces of the query Q.

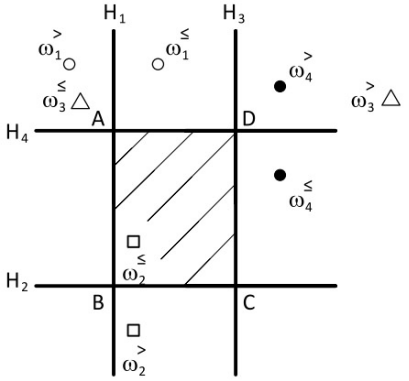


Fig. 4. Straight Rectangle ABCD

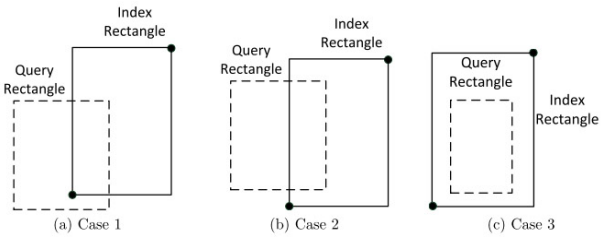
3.3. Our scheme

Our approach to EhQ work in the following way. In order to encrypt a request $Q = H_1^{\leq} \cap H_2^{\leq} \cap \ldots \cap H_{2d}^{\leq}$, we create anchors $\omega_i^{\leq}$ and $\omega_i^{>}$ for each hyperplane H_i. Then we generate an encrypted discriminator Δ_{H_i} for each H_i. Using Δ_{H_i}, by encryption we can determine, if this encrypted point V is present in $H_i^{\leq}$ or in $H_i^{>}$.

3.3.1. Algorithms for encrypting the range of vertices and queries

Our method uses the following algorithms.

$\mathrm{Enc\_Vertex}(V, M) \to [V]$. The data owner uses this algorithm to encrypt the vertex V of MBR node of $\hat{R}$-tree, using its secret key M, an invertible matrix $(d+1)\times(d+1)$.



(a) Case 1 (b) Case 2 (c) Case 3

Fig. 5. Three cases of intersection of rectangles

At the set vertex $V = (v_1, v_2, ..., v_d)^T$ the algorithm first adds an additional dimension, in order to create $V_+ = (V^T \mid 1)^T$. V is encrypted to $[V] = r_1 M^{-1} V_+$, where r_1 is a random positive number.

$\mathrm{Gen\_Anchor}(H) \to (\omega_i^{\leq}, \omega_i^{>})$. This algorithm takes a given hyperplane H, defined by parameters a and b and outputs reference points $\omega_i^{\leq}$ and $\omega_i^{>}$, located on $H^{\leq}$ and $H^{>}$ correspondently. It randomly chooses point $\omega_i^{\leq} \in H^{\leq} - H$ and calculates $\omega^{>}$ as reflected in H in the following way. If $\omega^{\leq}$ and $\omega^{>}$ are vectors, representing $\omega^{\leq}$ and $\omega^{>}$ correspondently, we require, their vector difference $\omega^{\leq} - \omega^{>}$ was orthogonal to H. From linear algebra we know that vector $\mathbf{a}$ orthogonal to H. Let $\mathbf{a}^T \omega^{\leq} - b = \delta$. We have $\mathbf{a}^T \omega^{>} - b = -\delta$, $\mathbf{a}^T (\omega^{\leq} - \omega^{>}) = 2\delta$. As we know $\mathbf{a}$ and $\omega^{\leq}$, we can get $\omega^{>} = \omega^{\leq} - \frac{2\delta}{\| \mathbf{a} \|^2} \mathbf{a}$.

$\mathrm{Gen\_Discr}(\omega^{\leq}, \omega^{>}) \to \Delta_H$. This algorithm receives reference points $\omega^{\leq} = (\omega_1^{\leq}, \omega_2^{\leq}, ..., \omega_d^{\leq})^T$ and $\omega^{>} = (\omega_1^{>}, \omega_2^{>}, ..., \omega_d^{>})^T$, corresponding hyperplanes H, and outputs discriminator Δ_H. First, it adds information about the distance to the reference points to get $\omega_+^{\leq} = ((\omega^{\leq})^T \mid (-0.5 \| \omega^{\leq} \|^2))^T$ and $\omega_+^{>} = ((\omega^{>})^T \mid (-0.5 \| \omega^{>} \|^2))^T$. Further $\omega_+^{\leq}$ and $\omega_+^{>}$ are encrypted using M as $\langle \omega^{\leq} \rangle = M^T \omega_+^{\leq}$ and $\langle \omega^{>} \rangle = M^T \omega_+^{>}$. Finally, the algorithm chooses a random positive value r_2 and generates an encrypted discriminator $\Delta_H = r_2 (\langle \omega^{\leq} \rangle - \langle \omega^{>} \rangle)$.

3.3.2. Halfspace range queries in encrypted MBR vertices

The cloud performs an encrypted half-space range query in the encrypted MBR vertices using the following algorithms.

$\text{Halfspace\_Qry}([V],\Delta_H) \to V_H^{\leq}$. This function takes a set of encrypted vertices MBR [V] and hyperplane discriminator Δ_H and shows a dataset $V_H^{\leq} = V \cap H^{\leq}$.

It works by calling the following function to test each point in [V].

$\text{In\_Halfspace}([V],\Delta_H) \to \{0,1\}$. The function accepts encrypted point [V], discriminator Δ_H and outputs a bit indicating whether $V \in H^{\leq}$ calculates $\Delta_H \cdot [V]$. As

$$\Delta_H \cdot [V] = r_1 r_2(\langle\omega^{\leq}\rangle - \langle\omega^{>}\rangle)\cdot[V] =$$
$$= r_1 r_2((M^T\omega_+^{\leq}) - (M^T\omega_+^{>})^T M^{-1} V_+ =$$
$$= r_1 r_2(\omega_+^{\leq} - \omega_+^{>})^T V_+ =$$
$$= r_1 r_2(\| \omega_+^{\leq} - V \|^2 - \| \omega_+^{>} - V \|^2)^T V,$$

$\Delta_H \cdot [V] \geq 0$, if V is located in $H^{\leq}$. Function outputs 1, if V is located in $H^{\leq}$ and 0 otherwise.

3.3.3 Intersection of a hyperrectangle

We show how to determine the intersections between encrypted d-dimensional query and hyperrectangles of the index based on queries of the halfspace range. We require that each surface of the hyperrectangle is orthogonal to the coordinate axis. That is, each face is a hyperplane $H_i = (x_1,...,x_{i-1},c_i,x_{i+1},...,x_d)$, where c_i is constant, but x_i are not limited. This restriction is necessary, since intersection tests using half-space queries may not work on common polyhedra, as we shall see.

Hyperrectangle index $R \subset \mathbf{R}^d$ now it is completely determined by its extreme vertices $V_\perp, V_\top \in R^d$, defined as follows. If $V = (v_1, v_2,..., v_d)$ is the vertex R, then we define $V_\perp = (\min\{v_1\},...,\min\{v_d\})$, $V_\top = (\max\{v_1\},...,\max\{v_d\})$, where min and max are taken over all the vertices V from R.

However, the hyper-rectangle of the query is given in terms of half-spaces defined by its faces, since $Q = H_1^{\leq} \cap H_2^{\leq} \cap ... \cap H_{2d}^{\leq}$.

The hyperrectangle of the index is encrypted as follows. $\text{Enc\_Index}(R,M) \to [R]$. Taking into account hyperrectangle of index $R = (V_\perp, V_\top)$ and key matrix M this algorithm outputs encryption R as $[R] = ([V_\perp],[V_\top])$, $[V_\perp] = \text{Enc\_Vertex}(V_\perp, M)$ and $[V_\top] = \text{Enc\_Vertex}(V_\top, M)$.

$\text{Enc\_Query}(Q,M) \to Q$. This algorithm takes a key matrix M and the query area Q, given as the intersection of half-spaces $H_1^{\leq} \cap H_2^{\leq} \cap ... \cap H_{2d}^{\leq}$. For each H_i it firstly invokes $\text{Gen\_Anchor}(H_i)$, in order to get $\omega_i^{\leq}$ and $\omega_i^{>}$. Then it gets Δ_{H_i}, invoking $\text{Gen\_Discr}(\omega_i^{\leq}, \omega_i^{>})$. It returns the area of encrypted request $\langle Q\rangle = (\Delta_{H_1}, \Delta_{H_2},..., \Delta_{H_{2d}})$.

$\text{Xsect\_Index}([R],Q) \to \{0,1\}$. This function accepts hyperrectangles of encrypted query and index. It outputs logic value, indicating, whether the hyperrectangles intersect. If both $V_\perp$ and $V_\top$ are defined as lying outside $H_i^{\leq}$ for some H_i, otherwise algorithm returns 0 and 1 (look Algorithm 1).

Algorithm 1. Xsect_Index

input: $[R] = ([V_\perp],[V_\top])$, $\langle Q\rangle = (\Delta_{H_1},...,\Delta_{H_{2d}})$

output: {0,1}

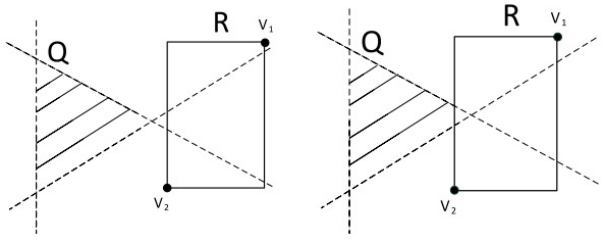
1 **for each** $\Delta_{H_1} \in \langle Q\rangle$ **do**

2 **if not** $\text{In\_Halfspace}([V_\perp],\Delta_{H_1})$, **and not** $\text{In\_Halfspace}([V_\top],\Delta_{H_1})$

then
return 0
4 **end**
5 **end**
6 **return** 1

3.3.4. Polyhedral query domains

Our scheme can handle arbitrary convex polyhedral query areas, but can introduce false positives. The two cases shown in fig. 6, cannot be distinguished.



(a) No intersection (b) Intersection

Fig. 6. Half-space queries using a polyhedral domain

Both cases return the same results if we run half-space queries for vertices of the hyper-rectangle index, using $H_i^{\leq}$, defining the region of triangular queries. However, our scheme is safe for convex polyhedral domains of the query, since it does not introduce false negations in accordance with Theorem 1.

Theorem 1. $\text{Xsect\_Index}([R],\langle Q\rangle)$ runs 0, if the convex domain of a polyhedral query Q and the range of indices R do not intersect.

Proof. R is defined by the formula $V_\perp = (\min\{v_1\},...,\min\{v_d\})$ and $V_\top = (\max\{v_1\},...,\max\{v_d\})$, its extreme vertices. If in

$\text{In}\_\text{Halfspace}([V_{\perp}], \Delta_{H_1})$ and $\text{In}\_\text{Halfspace}([V_{\top}], \Delta_{H_1})$ development 0 for a half-space query $\Delta_{H_1} \in \langle Q \rangle$, then neither $V_{\perp}$ nor $V_{\perp}$ are not located in $H^{\leq}$. Since $V_{\perp}$ and $V_{\perp}$ have, correspondently, the smallest and largest projection along the axis i, then any 2^d-vertices in R cannot be in $H^{\leq}$. It is clear, that Q and R do not intersect.

3.3.5. Construction and query of $\hat{R}$-tree

Tree $\hat{R}$ can be considered as R-tree, which MBR are encrypted, but the relationship of parents and children - not. Its data points are encrypted separately. These encrypted data points, encrypted indexes of $\hat{R}$-tree and the parent-child relationship are placed to the cloud. We check the overlap between encrypted query ranges and encrypted MBR trees using a new mechanism Encrypted Half-space Query (EhQ).

Our approach EhQ is safe and effective and can be used for search under the encryption of other complex data structures, such as BNL-trees and kD-trees. We decided to set up our index of $\hat{R}$-tree on R-tree, because family of R-trees has more lower information lack, then bouquet schemes, k-tools, BNL-tree, kD-tree and etc., as shown in [3].

$\hat{R}$-tree constructed as in the algorithm 2. Let S and $\tilde{S}$ denote encrypted versions of the data set. For MBR leaf let SR means data points falling into R, and $\tilde{S}_R$ denote encrypted texts S_R. Let $\mathbf{T}$ and $\hat{\mathbf{T}}$ denote R-tree and correspondent $\hat{R}$-tree correspondently. Let $\mathbf{PC}$ indicates a set of parent-child relations in $\mathbf{T}$.

The following function, developed in Algorithm 3, finds all the leaves of a tree $\hat{R}$, which intersect the specified range request.

$\hat{\mathbf{R}} - \text{tree}\_\text{Qry}\left(Q, \hat{\mathbf{T}}\right) \to \mathbf{L}$ This function accepts the input query $\langle Q \rangle$ of encrypted range of $\hat{R}$-tree $\mathbf{T}$. Its output is a set of encrypted $\mathbf{L}$ leafs, which MBR intersect $\langle Q \rangle$.

3.4. Security analysis

Our analysis shows that outsourcing schemes that eliminate the simple order of encryption of tuples cannot provide reliable guarantees of confidentiality. Such information about the order often allows the enemy to accurately estimate the values of encrypted tuples. Let's start with the security analysis of the scheme used to encrypt the hypersurfaces of the index and the query in $\hat{R}$-trees. Then we compare the confidentiality guarantees provided by our scheme to those provided by competing schemes, especially when the enemy was able to detect partial information about the values of encrypted tuples.

Algorithm 2. Construction of $\hat{R}$-tree

input: $\mathbf{T}$, M
output: $\hat{\mathbf{T}}$
$\hat{\mathbf{T}} = \varnothing$
$\mathbf{PC} = \varnothing$ $stack = \varnothing$
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.R denotes node's MBR
[R] = Enc_Index(node.R, M))
, **if** node has children **then**
for each child **do**
Save the parent-child relationship to **PC**
end
end
if node is not a leaf **then**
for each child **do**
stack.Push(child)
end
// Generate a node for $\hat{\mathbf{T}}$
onode={[R]}
else
Encrypt data points in S_R to obtain $\tilde{S}_R$
onode={[R], $\tilde{S}_R$}
end
Add onode to $\hat{\mathbf{T}}$
end
Add **PC** to $\hat{\mathbf{T}}$
return $\hat{\mathbf{T}}$

Algorithm 3. $\hat{R}$-tree Qry

Input: $\langle Q \rangle$, $\hat{\mathbf{T}}$
$\mathbf{L} = \varnothing$
$stack = \varnothing$
node = $\hat{\mathbf{T}}$.root
if Xsect_Index(node.[R], $\langle Q \rangle$) **then**
stack.Push(node)
end
else
return $\varnothing$
end
while $stack \neq \varnothing$ **do**
node = stack.Pop())

Algorithm 2. Construction of $\hat{R}$-tree
input: **T**, M
output: $\hat{\mathbf{T}}$
$\hat{\mathbf{T}}=\varnothing$
PC $=\varnothing$ *stack* $\neq\varnothing$
node = **T**.root
if node $\neq$ NULL **then**
stack.Push(node)
end
else
return $\varnothing$
end
while *stack* $\neq\varnothing$ **do**
node = stack.Pop()
// node.R denotes node's MBR
[[R] = Enc _ Index(node.R, M))
, **if** node has children **then**
for each child **do**
Save the parent-child relationship to **PC**
end
end
if node is not a leaf **then**
for each child **do**
stack.Push(child)
end
// Generate a node for $\hat{\mathbf{T}}$
onode={[R]}
else
Encrypt data points in S_R to obtain $\tilde{S}_R$
onode={[R], $\tilde{S}_R$}
end
Add onode to $\hat{\mathbf{T}}$
end
Add **PC** to $\hat{\mathbf{T}}$
return $\hat{\mathbf{T}}$

Algorithm 3. $\hat{R}$-tree Qry
Input: ⟨Q⟩, $\hat{\mathbf{T}}$
L $=\varnothing$
stack $\neq\varnothing$
node = $\hat{\mathbf{T}}$.root
if Xsect _ Index(node.[R], ⟨Q⟩) **then**
stack.Push(node)
end
else
return $\varnothing$
end
while *stack* $\neq\varnothing$ **do**
node = stack.Pop())
if node is a leaf **then**
L = **L** $\cup$ node
end
else
for each node's child **do**
if Xsect _ Index(child.[R], ⟨Q⟩) **then**
stack.Push(child)
end
end
end
return **L**

Table 1

Encrypted database schemas. N - number of tuples, C – bouquet size. (Maximum overheads logN [2; 4] can be achieved only for one-dimensional data)

Scheme	Query Overhead	Reveals Order?
Bucketization [3]	**High**: O(N / C)	No.
Order preserving [2]	Low: logN	**Yes**.
Predicate encryption [4]	Low: logN	**Yes**.

3.4.1. Encryption Security

We encrypt the hyper-rectangles of the index and the query using ASPE, which was proven to be safe for open-text attacks in [10]. Our schema stores the ASPE security properties, because it extends ASPE, but does not change the basic approach to encryption in [10]. Artificial dimensions and random asymmetric splitting still work in our scheme.

3.4.2. Comparison with competing schemes

We show how information about the order can be used by the enemy to output plaintext values using order statistics. As shown in tabl. 1, the operating schemes provide either efficiency or privacy protection, but not both. In this respect, bouquet scheme from [3] protects information about the order, but suffers because of the high overhead of the request. It also requires the data owner to manage the bouquet indexes. Otherwise, [2; 4] allow efficient queries, but display information about the order on the encrypted tuples. $\hat{R}$-tree can perform very efficient queries, hiding the order of data points in each MBR sheet.

For several reasons, we will not make a detailed comparison of our scheme with the bouquet scheme [3]. At first, [3] is not a true out-sourcing scheme, since the index is stored on the data owner's site, and not in the cloud. It also requires that all requests be made by the data owner, and this means significant requirements. Finally, the index search takes time $O(N/C)$, which is linear in the size of the database, if we keep the size of the bouquet constant. These overheads are excessive compared to competing schemes.

Therefore, we compare the stability of our scheme with the ability of schemes in [2; 4]. This is a suitable

comparison, because $\hat{R}$-tree achieves the same complexity of queries as these schemes. We will show that our scheme has much better stability, the advantage it has against any scheme that does not hide the ordering of tuples.

3.4.3. Attack Model

Let A_O denotes the detractor in schemes that reveal information for ordering, and let $A_{\hat{R}}$ denote the detractor in our scheme. Ranges of leaf indices of $\hat{R}$-tree are the limiting fields for clusters of encrypted data points. The nodes of a higher level are additional clusters of bounding blocks. $A_{\hat{R}}$ cannot see the ranges of indexes in any of the nodes $\hat{R}$, because they are encrypted using ASPE. Nevertheless, $A_{\hat{R}}$ is able to study the ordering of all MBR leaf from a sufficient number of query results. $A_{\hat{R}}$ can selectively take a half-space discriminator Δ_{H_i} from received requests for the formation of new requests. But these requests can help him to get a list of MBR. The order of location of data points within each leaf MBR is still safe.

The aim $A_{\hat{R}}$ is to output the values of the encrypted data points belonging to the node λ_j of list. $A_{\hat{R}}$ can examine the values of plain text of some encrypted data points. Assume, that $A_{\hat{R}}$ knows the lower and upper limits of the range in λ_j, and he also knows the distribution of the values of points.

To compare our schema with methods such as [2; 4], which show the order of the encrypted points, we assume that A_O also tries to output the values of the encrypted data points in λ_j. A_O knows the relative order of all data points, the lower and upper limits of the range in λ_j and distribution of points values.

3.4.4. Optimal evaluation criterion of the detractor

The goal of detractor is to deduce the values of the encrypted tuples. With this aim it will use a statistical evaluation, which effectiveness should be measured in terms of shown error. We will use wide Mean Square Estimation Error (MSEE), also used in [3], which is working as follows. For simplicity, we consider one-dimensional case.

It is often necessary to estimate the value of a random variable Y, which is inaccessible, in terms of $g(X)$ function, available random variable X. In our case Y is the plain text of the tuple. The detractor selects a suitable random variable X. MSEE is determined as $\mathbf{E}[(Y-g(X))^2]$. The simplest choice for the detractor is $g(X)=c$, a constant. We find the value c_{min}, which minimizes MSEE as follows. Starting from

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

we differentiate along c and establish in 0, get $c_{min} = \mathbf{E}[Y]$. MSEE now is $\mathbf{E}[(Y-\mathbf{E}[Y])^2 = \mathrm{Var}(Y)$. Therefore, the optimal estimate for Y is $\mathbf{E}[Y]$, which reaches the minimum value of MSEE $\mathrm{Var}(Y)$.

Therefore, considering the encrypted tuple $\tilde{y}_i$ with open text y_i, taken from the distribution, modelling by a random variable Y, the best estimation, which the detractor can make for y_i, is equal to $\mathbf{E}[Y]$, achieving MSEE $\mathrm{Var}(Y)$.

3.4.5. Attacks on installation with information about the order and without it

Given the continuous range $R=[y_s, y_e]$, containing $|R|$, encrypted tuples $(\tilde{y}_1, \tilde{y}_2, ..., \tilde{y}_{|R|})$, both detractors A_O and $A_{\hat{R}}$ try to show the meanings of open text y_s, y_e encrypted tuples in R. We assume that both A_O and $A_{\hat{R}}$ know the meaning of plaintext y_s, y_e from two endpoints of the range R. Let the random variable Y corresponds to the same distribution as the plaintext values of all encrypted tuples having a density function $f(y\}$. Besides, A_O knows how distribution $f(y\}$ of open texts y_i, as well as the order of encrypted sets $\tilde{y}_i$. $A_{\hat{R}}$ know distribution $f(y\}$, but not the order of encrypted tuples $\tilde{y}_i$.

3.4.5.1. Attack $A_{\hat{R}}$ (order is unknown)

Let the random variable Y_R coincides with the same distribution as the values of the plaintext of encrypted tuples in R. Using the distribution $f(y)$, $A_{\hat{R}}$ finds distribution $f_{Y_R}(y)$ for Y_R. In p. 3.4.4. we have seen, that the best estimation $A_{\hat{R}}$ for any $\tilde{y}_i \in R$ - $\mathbf{E}[Y_R]$.

3.4.5.2. Attack A_O (the order is known)

A_O can do much better, as it knows the order $\tilde{y}_i$. A_O first finds $f_{Y_R}(y)$. Let random variable $Y_{(k)|R}(y)$ represents the value of plaintext k of The smallest set in the range R, having distribution $f_{(k)|R}(y)$. A_O gets $f_{(k)|R}(y)$, using $f_{Y_R}(y)$ and equation (1). Let $\tilde{y}_{(k)}$ mean k the smallest set in R. As in p. 3.4.4, the best estimation of A_O for $\tilde{y}_{(k)}$ open text $y_{(k)}$ is $\mathbf{E}[Y_{(k)|R}]$.

3.4.6. Metrics ε of absolute estimation error

Let $A_{\hat{R}}$ and A_O evaluate the true value of plaintext y_i for encrypted tuple $\tilde{y}_i \in R$, as $y_i^{\hat{R}}$ and y_i^O correspondently. Define the Absolute estimation Error

for $A_{\hat{R}}$ as $\varepsilon_{y_i}^{\hat{R}} = | y_i - y_i^{\hat{R}} |$ and for A_O - $\varepsilon_{y_i}^{O} = | y_i - y_i^{O} |$. If $\tilde{y}_{(k)}$ the smallest set in R, define $\varepsilon_{(k)}^{\hat{R}} = | y_{(k)} - \mathbf{E}[Y_R] |$ and $\varepsilon_{(k)}^{O} = | y_{(k)} - \mathbf{E}[Y_{(k)R}] |$.

Conclusions

This work represents $\hat{R}$-tree - hierarchical encrypted index, which can provide safe and effective bandwidth requests over encrypted data. $\hat{R}$-tree hides the order of internal MBR-files for privacy protection. Our theory and empirical analysis show that revealing order is dangerous for external data, and $\hat{R}$-tree has much better stability than the scheme without information protection ordering. Also the system, realizing $\hat{R}$-tree, having good capacity is developed.

References

1. Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D. and Rabkin, A. (2010), A view of cloud computing, *Communications of the ACM*, No. 53(4), pp. 50-58. https://dx.doi.org/10.1145/1721654.1721672.

2. Boldyreva, A., Chenette, N., Lee, Y. and O'Neill, A. (2009), Order-Preserving Symmetric Encryption. *Advances in Cryptology - EUROCRYPT 2009*, Springer Berlin Heidelberg, pp. 224-241. https://dx.doi.org/10.1007/978-3-642-01001-9_13.

3. Hore, B., Mehrotra, S., Canim, M. and Kantarcioglu, M. (2011), Secure multidimensional range queries over outsourced data, *The VLDB Journal*, No. 21(3), pp. 333-358. https://dx.doi.org/10.1007/s00778-011-0245-7.

4. Yanbin, Lu. (2012), *Privacy-preserving logarithmic-time search on encrypted data in cloud*.

5. Yiu, M.L., Ghinita, G., Jensen, C.S. and Kalnis, P. (2009), Outsourcing Search Services on Private Spatial Data, *25th International Conference on Data Engineering (ICDE), IEEE.* https://dx.doi.org/10.1109/ICDE.2009.185.

6. David, H.A. and Nagaraja, H.N. (2003), *Order Statistics*, third edition. Wiley, New York.

7. Katz, Jonathan, Sahai, Amit, and Waters, Brent (2008), Predicate encryption sup-porting disjunctions, polynomial equations, and inner products, *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology, EUROCRYPT'08*, pp. 146-162.

8. Li, M., Yu, S., Cao, N. and Lou, W. (2011), Authorized Private Keyword Search over Encrypted Data in Cloud Computing, *31st International Conference on Distributed Computing Systems (ICDCS). IEEE.* https://dx.doi.org/10.1109/ICDCS.2011.55.

9. Shen, E., Shi, E. and Waters, B. (2009), Predicate Privacy in Encryption Systems, *Theory of Cryptography*, Springer Berlin Heidelberg, pp.457-473. https://dx.doi.org/10.1007/978-3-642-00457-5_27.

10. Wai Kit Wong, David Wai-lok Cheung, Ben Kao and Nikos Mamoulis (2009), Secure Knn Computation On Encrypted Databases, *35th SIGMOD international conference on Management of data*, SIGMOD '09, pp. 139-152.

Надійшла до редколегії 15.02.2018
Схвалена до друку 20.03.2018

Відомості про автора:

Оваід Сальман Рашід
кандидат технічних наук
коледж університету Al Maaref,
Анбар, Республіка Ірак,
https://orcid.org/0000-0002-1189-9707
e-mail: owaidsalman@yahoo.com

Information about the author:

Owaid Salman
Doctor of Philosophy
Al Maaref University College,
Al Anbar, Iraq
https://orcid.org/0000-0002-1189-9707
e-mail:owaidsalman@yahoo.com

БЕЗПЕКА ЗАШИФРОВАНИХ БАЗ ДАНИХ В ХМАРІ

Сальман Рашид Уайд

Існуючі методи шифрування забезпечують або безпеку, або ефективність. Так, багато схем навіть розкривають порядок зашифрованих кортежів, що дозволяє досить точно оцінювати значення відкритого тексту. Для усунення цього недоліку в роботі представлено дерево (ієрархічний зашифрований індекс), який може бути надійно поміщений в хмару і ефективно спотворений. Ієрархічний зашифрований індекс заснований на механізмі, який був розроблений для зашифрованих запитів з використанням методу шифрування Asymmetric scalar product preserving encryption (ASPE).

Ключові слова: *шифрування запитів, шифрування баз даних, хмарні обчислення, ієрархічний індекс.*

БЕЗОПАСНОСТЬ ЗАШИФРОВАННЫХ БАЗ ДАННЫХ В ОБЛАКЕ

Сальман Рашид Уайд

Существующие методы шифрования обеспечивают либо безопасность, либо эффективность. Так, многие схемы даже раскрывают порядок зашифрованных кортежей, что позволяет достаточно точно оценивать значения открытого текста. Для устранения этого недостатка в работе представлено дерево (иерархический зашифрованный индекс), который может быть надежно помещен в облако и эффективно искажен. Иерархический зашифрованный индекс основан на механизме, который был разработан для зашифрованных запросов с использованием метода шифрования Asymmetric scalar product preserving encryption (ASPE).

Ключевые слова: *шифрование запросов, шифрование баз данных, облачные вычисления, иерархический индекс.*