available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

A schema for protecting the integrity of databases

Ibrahim Kamel

Department of Electrical and Computer Engineering, University of Sharjah, UAE

ARTICLE INFO

Article history:
Received 28 February 2009
Received in revised form
8 April 2009
Accepted 9 April 2009

Keywords:
Data integrity
Watermarking
Database
Spatial indexing
Multidimensional indexing

ABSTRACT

Unauthorized changes to databases can result in significant losses for organizations as well as individuals. Watermarking can be used to protect the integrity of databases against unauthorized alterations. Prior work focused on watermarking database tables or relations. Malicious alteration cannot be detected in all cases. In this paper we argue that watermarking database indexes in addition to the database tables would improve the detection of unauthorized alterations. Usually, each database table in commercial applications has more than one index attached to it. Thus, watermarking the database table and all its indexes improve the likelihood of detecting malicious attacks. In general, watermarking different indexes like R-trees, B-trees, Hashes, require different watermarking techniques and exploit different redundancies in the underlying data structure. This diversity in watermarking techniques contributes to the overall integrity of the databases.

Traditional relational watermarks introduce some error to the watermarked values and thus cannot be applied to all attributes. This paper proposes a novel watermarking scheme for R-tree data structures that does not change the values of the attributes. Moreover, the watermark does not change the size of the R-tree. The proposed technique takes advantage of the fact that R-trees do not put conditions on the order of entries inside the node. In the proposed scheme, entries inside R-tree nodes are rearranged, relative to a "secret" initial order (a secret key), in a way that corresponds to the value of the watermark.

To achieve that, we propose a one-to-one mapping between all possible permutations of entries in the R-tree node and all possible values of the watermark. Without loss of generality, watermarks are assumed to be numeric values. The proposed mapping employs a numbering system that uses variable base with factorial value.

The detection rate of the malicious attacks depends on the nature of the attack, distribution of the data, and the size of the R-tree node. Our extensive analysis and experimental results showed that the proposed technique detects data alteration with high probability (that reaches up to 99%) on real datasets using reasonable node sizes and attack model. The watermark insertion and extraction are mainly main memory operations, and thus, have minimal effect on the cost of R-tree operations.

1. Introduction

Information security and integrity are becoming critical areas of research especially with the recent outburst of Internet attacks and hostility. Adversaries launch attacks for many reasons, most importantly, profit. Databases usually contain critical information like salaries, ownership, land use, personal information, etc. Unauthorized changes to such databases might result in significant losses for organizations and individuals. Recently database research community

E-mail address: kamel@sharjah.ac.ae

0167-4048/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2009.04.001

realized the importance of watermarking in protecting relational databases and especially databases published on the web (Agrawal et al., 2003; Gross-Amblard, 2003; Guo et al., 2006; Iyer et al., 2004; Li et al., 2003; Qin et al.; Sion et al., 2003b).

The state of the art in detecting unauthorized alterations in relational databases can only detect some of the attacks on specific types of attributes. These techniques are using probabilistic models, and thus, the detection rate depends on the type and the value of the modification. Moreover, prior database watermarking techniques are designed to protect specific types of data, e.g., attributes that tolerate distortion to their values.

In this paper we advocate improving the detection rate of malicious alteration by watermarking not only the relational tables (data records) but also all relevant indexes. Database tables usually have multiple indexes per table. This paper proposes a fragile watermarking technique for protecting data integrity in databases and more specifically in R-tree data structures. R-trees are used in indexing multidimensional data, e.g., spatial databases or GIS where maps, properties, and land use are stored. Malicious user might attempt to change the location or the size of data objects. An example for a possible attack in this case is increasing the size of a piece of property. The proposed watermarking algorithm detects, with high probability, unauthorized changes in the data.

R-trees can also be used to index multiple attributes in traditional databases, e.g., indexing the age and salary attributes in ''Employee'' relation. In this case, each employee is represented in the R-tree by a two dimensional point representing his/her age and salary. Malicious user might attempt to change the salary value, for example, and as a result the corresponding point changes its position in the two dimensional space.

The proposed technique should not be perceived as a tool to only protect stand alone R-tree. Indexes, like R-trees, are usually integrated into the database management systems (DBMS). The proposed R-tree watermarking technique is expected to be integrated in the DBMS and to complement other relational watermarking techniques to provide better data integrity to the databases. In general, enterprise applications consist of key components like: database server, application servers, and web front-end. Each of these components has its own vulnerabilities. Examples of such vulnerabilities are:

- Weak and default password,
- Buffer overflow,
- Miss-configurations, and
- Resource privilege management.

Even though the R-tree will be accessed by the DBMS, yet the R-trees as well as the database tables are still vulnerable to attacks by unauthorized users or malicious code. Attacker might exploit one or more of the above vulnerabilities to change the values of attributes in the database. Moreover, the fact that the database relations and their indexes are disk resident, makes them vulnerable to attacks by third party code even if the DBMS is not running (off-line attack).

Most of the watermarking techniques developed to protect tables (relations) (Agrawal et al., 2003; Guo et al., 2006), on the other hand, are not suitable for attributes like salaries and property limits, because they introduce distortions to the watermarked attributes. Even with database relations that are not sensitive to distortion, the traditional watermarking techniques do not always detect the malicious alteration. Thus, watermarking various database indexes (in addition to database tables) would strengthen the overall security of the database.

The use of secure hashing like MD5 or SHA and digital signature techniques for integrity protection in R-trees requires large storage overhead and incurs heavy performance penalty as explained in Section 4. The proposed watermarking algorithm rearranges the entries in the R-tree node, with respect to a predetermined reference order, according to the value of the watermark. The watermark value and the reference order are secret and known only to the database owner, and thus, unauthorized updates corrupt the watermark. Later in the paper we show that unauthorized updates can be detected with high probability. This type of watermark is known as fragile watermark, which is used for maintaining data integrity. In (Kamel and Albluwi, 2009) we introduced a robust watermarking technique for R-tree to protect the copyright of software code. Robust watermark significantly differs from fragile watermarking; it is expected to withstand malicious attacks and protect the secrete message. Unlike robust watermarks, fragile watermarks get corrupted if the data (or the cover) is changed.

R-trees are widely used for indexing spatial data (Manolopoulos et al., 2005), Spatio-temporal databases (Sun et al., 2006), etc. Protecting disk-based data structures, like R-trees, B-trees, is more challenging than memory-based data structures, like binary tree and graphs. The reason is that memory-based data structures can be attacked only during the program execution whereas disk-based data structures can be attacked off-line even if the program is not running.

The proposed watermarking technique has the following desirable features:

- It does not change the values of the data in the R-tree node but rather hides the watermark in the relative order of entries inside the R-tree node.
- It does not increase the size of the R-tree.
- The proposed technique does not interfere with R-tree operations.
- The performance overhead is minimal.
- The integrity check does not require the knowledge of un-watermarked data (blind watermark).

In the next section, we explain the proposed method for watermarking R-trees and the watermark insertion algorithm. Section 3 explains the attack detection algorithm and how we can improve the detection rate at the expense of time and space overhead. In Section 4, we present simulation experiments that show the effectiveness of the proposed technique and measure the incurred overhead. Prior works in watermarking databases are summarized in Section 5. Concluding remarks and future work are presented in Section 6.

2. Data integrity in R-trees

R-tree (Guttman, 1984) is the extension of the B-tree for multidimensional objects. For simplicity, this discussion uses data in two dimensional space; however, R-tree and its variants work for any number of dimensions. A geometric object is represented by its minimum bounding rectangle (MBR). Non-leaf nodes contain entries of the form (*ptr*, *R*) where *ptr* is a pointer to a child node in the R-tree; *R* is the MBR that covers all rectangles in the child node. Leaf nodes contain entries of the form (*obj-id*, *R*), where *obj-id* is a pointer to the object description, and *R* is the MBR of the object. R-trees allow nodes to overlap. This way, the R-trees can guarantee at least 50% space utilization and at the same time remain balanced. Each node in the tree (except the root) contains between m (minimum number of entries per node) and M (maximum number of entries per node) entries, where $m = M/2$. At the same time, each non-leaf node (except the root node) has between m and M child nodes. R-tree is a balanced tree; meaning all leaves appear on the same level.

The tree grows bottom-up; when a node overflows, the split routine is invoked to split the node into two. Search, insert, delete, split and merge are the popular R-tree operations. The most frequent operation is search. Many techniques have been developed to improve its performance (Beckmann et al., 1990; Kamel and Faloutsos, 1994). Contrary to the B-tree, R-trees do not put condition on the order of entries inside the node. Thus, the search algorithm inspects all entries in the node. Our watermarking algorithm takes advantage of this feature, and hides the watermark by changing the order of entries in the tree. Our proposed technique works equally on other R-tree variants e.g., packed R-tree (Kamel and Faloutsos, 1993), Hilbert R-tree (Kamel and Faloutsos, 1994), and the R*-tree (Beckmann et al., 1990).

2.1. *R-tree watermarking: the basic idea*

The main objective is to identify unauthorized updates to the R-tree. Authorized updates are watchful for the hidden watermark and update it as needed. The watermark is hidden in the set of entries (or MBRs) by re-arranging them inside the R-tree node in a specific way that corresponds to the value of the watermark *W*. This re-arrangement is done relative to an initial secret reference order. An example of the initial reference order might be to sort the entries relative to the *x* value of the lower left corner of the MBR (called *Lowx*).

Fig. 1 shows a set of MBRs[1] that is stored in one R-tree node. The figure highlights two objects *A* and *B* (shown with dark lines) and their *Lowx* points as shown as L_A and L_B respectively. In the sorted list of entries, MBR *B* appears before MBR *A* because $L_B < L_A$.

Given the watermarked set of entries and the value of watermark *W*, one can reconstruct the initial reference order of entries. Suppose that the Mallory (the adversary) increased the size of the object *A*; shown by dotted line in Fig. 1. The new

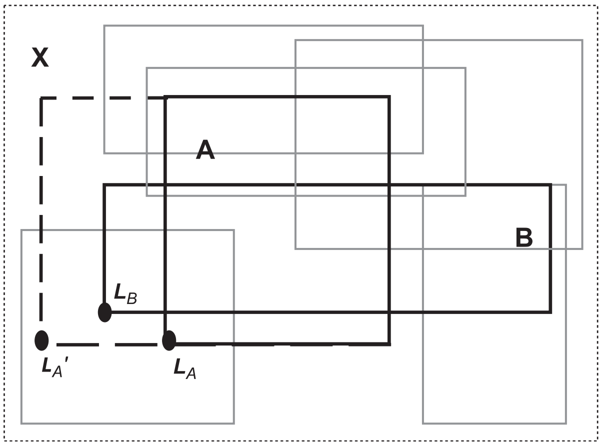

Fig. 1 – Attack on node *X*, the size object *A* increased by changing its *Lowx* value from L_A to L'_A.

location of *Lowx* of MBR *A* after Mallory's attack is L'_A comes before L_B. Now using the watermark value *W* the integrity check algorithm tries to reconstruct the initial reference order of this node, which is expected to show entry *B* before *A* in the secret order. However, object *A* should have appeared before object *B* because L'_A, the new *Lowx* of object *A* (after attack), is now smaller than L_B. This violation of initial reference order signifies that data has been attacked.

Obviously, the detection of the attack depends heavily on the criterion used in the reference order. Thus, if Mallory changed the y-axis only, we would not be able to detect the attack using *Lowx*. Section 2.4 discusses, in more details, other criteria that can be used in the reference order sorting. Also, if the attack was such that L'_A does not cross the L_B (or any other object) we would not be able to detect it. In Section 3.2 we show how to improve the detection rate for smaller alterations.

2.2. *Attack model*

Consider, for example, geographical information system where R-tree is used to index land uses and roads. Since databases are usually disk resident, Mallory can use a stand-alone code to illicitly change the data in the R-tree. The following are some possible attacks that Mallory can launch against the database:

Modification attack: increasing or decreasing the size of a region (MBR) in the node.
Displacement attack: change the location of a MBR without changing its size.
Insertion attack: insert a new MBR that did not exist in the original data.
Removal attack: remove an object, represented by an MBR, from the R-tree.

Notice that insertion and deletion attacks would be easier to detect as they will most likely disturb the order of the entries inside the R-tree node.

[1] In the rest of the paper, the terms 'entry' and 'MBR' are used interchangeably.

Table 1 – Notation table.

W	The value of the watermark
W_F	Watermark in factorial number system
E	A set of entries, where entries ordered as:$E = \{E_1, E_2, E_3 \ldots E_k\}$
E_R	The set of entries E sorted according to the secret reference order.
E_W	The set of entries E after embedding the watermark.

2.3. *Watermark insertion*

To be able to rearrange the entries of the node in a way that corresponds to a specific watermark value, we need to establish a one-to-one mapping between all possible permutations of the entries on one side and all values of W on the other side. A list of the symbols used in the rest of the paper is shown in (Table 1).

There are $k!$ different ways to arrange the entries $e_1, \ldots, e_k$. If W is an integer decimal number with d digits, then there are 10^d different possible values for W. The first problem we face is that the number of possible values of W represented in decimal format does not match the number of permutation $k!$. Moreover, There is no simple relationship between the value of k and the value of t, such that $10^t = k!$. Notice also that a single increment in the ith digit of W would result in a jump of 2^i in the value of W. This led us to believe that the use of the decimal numbering system does not lend itself naturally to one-to-one relationship with permutations. We worked out the requirements and details of a numbering system that fulfills the previous requirements. We realized the need for a numbering system with variable base. After developing a suitable numbering system, we found an unpopular numbering system that dated back to 1800s (Albluwi and Kamel, 2006; Smarandache, 2000) that is similar to ours, called *factorial numbering system*. Thus, the first step in the watermarking algorithm is to convert the value W decimal format to the *factorial* format (W_F) before embedding it in the R-tree.

2.3.1. *Factorial numbering system*

This system has variable base, meaning, the base of digit i is different from the base of digit j, $\forall\ i \neq j$ (on the contrary, the base of all digits in the binary system is always 2). The weight value of digit number i equals $i!$. Any integer can be represented as:

$$\sum_{k=1}^{n} [a k * k!]$$

a_k can take the values from 0 to k only; the least significant digit can take the values 0 or 1 while the 3rd digit can take the values 0, 1, 2, or 3. This is different from traditional numbering systems where each digit can take values from 0 to ($base - 1$). The integer $(859)_{10}$ can be represented as

$$\begin{aligned}(859)_{10} &= (1*1!) + (0*2!) + (3*3!) + (0*4!) + (1*5!) + (1*6!) \\ &= (110301)_{\text{factorial}}\end{aligned}$$

Based on the above, we can define *factorial numbering system* as: a number system where the base of the kth place is $k!$, and the allowed coefficients are between 0 and k. The restriction on the coefficients is necessary to make *one-to-one* mapping with the decimal system.

2.3.2. *Watermark embedding algorithm*

The embedding algorithm sorts entries in R-tree node according to a reference order E_R, which acts as a secret key and is known to the owner only. W_F is used to re-arrange the entries as follow: starting from the reference order E_R, we use left-circular shift operations to change the order of entries. First, we shift-left all entries a number of times equal to the most significant digit of W_F. Then, we freeze the left-most entry and shift-left the rest of entries a number of times that is equal to the value of the next most significant digit in W_F. Next we freeze the second left-most entry and so on and so forth. The watermark insertion algorithm is outlined in Fig. 2.

Let us clarify this by an example (shown in Fig. 3). Let us assume that there are four entries {Car, Bike, Scooter, Roller} in an R-tree node and W_F = "311". Moreover, let us assume that the "alphabetical order" is our initial reference order, thus, E_R = {Bike, Car, Roller, Scooter}. Since W_F has three digits, then the shuffling function is executed three times, as illustrated in Fig. 3. Since the first digit in W_F is 3, then during the first round, all entries will be shifted to the left three positions. After the first round, the new order will be {Scooter, Bike, Car, Roller}. In the second round, we will freeze entry 'Scooter' and process the other three entries {Bike, Car, Roller}. Since the second digit in W_F is '1' the subset {Bike, Car, Roller} will be shifted only once, resulting in {Scooter, Car, Roller, Bike}. In the third round, we will freeze the entry 'Car' (in addition to 'Scooter') and shift-left the subset {Roller, Bike} one time. The final order of the entries would be E_W = {Scooter, Car, Bike, Roller}.

2.4. *Sorting criterion for the reference order* E_R

The reference order E_R and the watermark value W are secret and are known only to authorized users. The significance of the reference order lies in the fact that it directly affects the sensitivity of the watermark to alterations. The reference order should be selected so that it would be sensitive to any of the above attacks. The following are some simple sorting criteria:

Lowx: sorts the node entries on the x value of the lower left corner of the corresponding MBR. Using the y value or any of

```
Algorithm Name: Watermark insertion
Input: E list of entries of size k;
        watermark W_F
Output: E_w
For (i = k-1 ; i > 0 ; i--)
//Circular-left-shift the subset E[x,y]
//by the value W_F [i]
        leftCircularShift(W_F[i] , E[i+1,k])
Where:
E[x,y] is a subset of E from x to y
W_F[i] is the i^th digit in W_F
```

Fig. 2 – Pseudo code for watermark insertion algorithm.

	Original sequence		Car	Bike	Scooter	Roller
	Reference Order		Bike	Car	Roller	Scooter
Round	**Order**	**Digit**	**Order after shifting**			
1	Most Significant Digit	3	Scooter	Bike	Car	Roller
2	Middle Digit	1	Scooter	Car	Roller	Bike
3	Least Significant Digit	1	Scooter	Car	Bike	Roller

Fig. 3 – Watermark insertion example.

the coordinates of the upper right corners would give the same results. *Lowx* is simple but it can detect attacks on one of the four sides of the MBR.

Area: entries in the node are sorted according to the area of the MBR, say, in ascending order. *Area* can catch insertion attacks, deletion attacks and attacks on the size of the object. However, it is blind to displacement attack.

Perimeter: sorts entries according to the value of the perimeter of the MBR. The sensitivity of the *Perimeter* is similar to *Area*.

Hash: this heuristic concatenates the four coordinates (in the 2-dimensional case) of the MBR and sorts the entries on the hash value of the concatenated string. In the performance evaluation (Section 4) we use MD5 hashing algorithm. *Hash* is sensitive to all four attacks mentioned in Section 2.2.

One can think of many other sorting criteria. The choice of the sorting criterion depends on several factors, e.g., the application, nature of the data, and the expected attack model.

3. Attack detection

The purpose is to identify attacked nodes in the R-tree if any. Authorized users run the integrity checking algorithm on an R-tree node either occasionally as a separate operation or before every read. Later in this paper, our experimental results (Section 4) show that the integrity checking algorithm is light weight and its cost can be ignored relative to the disk read cost. Since Mallory does not know the watermark value W nor the reference order E_R, changing data in R-tree nodes would corrupt the existing watermark.

3.1. *Integrity check algorithm*

The idea is that, given a watermarked node E_W and the secret watermark value W, we can reconstruct the initial reference order E_R. Extracting E_R from W and E_W is opposite to the watermark insertion algorithm shown in Fig. 2. The reference order E_R is expected to follow one of the criteria described in Section 2.4. Let us say, for example, that the reference order follows the *Area* sorting criterion. In this case, in the reconstructed E_R, the MBR with the smallest area should appear first; followed by the next larger MBR, and so on. However if the order of the entries in E_R does not follow the expected order (*Area* in this case), then, at least, one of the minimum bounding rectangles has been attacked (or changed). This is valid for any of the four types of attacks mentioned in Section 2.2. On the other hand, the opposite is not always true; if entries respect the order of *Area* does not necessarily mean that there were no attack. MBR might have been attacked but the change is too small to violate the *Area* sorting criterion. Thus, if the entries respect the expected order, this gives us confidence that data has not been attacked or the attack is not significant. The next section discusses how to improve the detection rate by using multiple sorting criteria.

Thus, in the design phase we should select a sorting criterion (for the reference order) that is sensitive to the smallest change that is considered significant by our application. Another factor that affects the detection of malicious attacks is the size of the R-tree node or the number of watermarked entries.

3.2. *Improving detection rate*

The ability to detect malicious attacks depends on the sensitivity of the E_R sorting criteria. Notice that some attacks might not be detected either because the reference order is not sensitive to this attack or the changes in the data are small to the extent that they did not disturb the reference order. To improve the detection rate of the proposed technique, we propose to use multiple watermarks with multiple reference orders.

For the simplicity of presentation, the case of two reference orders is described but the same concept can be extended to more than two reference orders. Recall that entries in E_W are physically arranged using a watermark value W relative to a secret reference order E_R; let the pair (W, E_R) be the *primary* watermark–reference pair. If we define a second hypothetical reference order E'_R then:

"Given the current arrangement E_W and the hypothetical reference order E'_R one can calculate the corresponding W' (called derived watermark) that relates E'_R to E_W".

The value W' is called the *derived watermark* because it is not chosen but rather calculated based on the chosen E'_R. The pair (W', E'_R) is called the *secondary* watermark–reference pair.

The use of two reference orders is expected to improve the attack detection rate because there is a chance that if the malicious alteration does not violate E_R it would violate the order E'_R. If E_R and E'_R are chosen carefully, then multiple reference orders would improve the detection of malicious attacks considerably as shown by simulation experiments in Section 4.5.6.

The insertion algorithm works as follow; let (W, E_R) and (W', E'_R) be the primary and secondary watermark–reference pairs, respectively.

Step1: Sort the entries inside the R-tree node according to the value W (with respect to the first reference order E_R) and calculate E_W (Fig. 4).

Step2: Use E_W and E'_R to calculate the corresponding *derived* watermark W'. The value of W' needs to be stored for each node. Thus, the owner of the database will have two watermarks: the *primary* watermark that he chose and a set of *secondary* watermarks that are calculated based on E'_R.

The malicious attack detection algorithm works as described in Section 3 except that it checks the validity of both

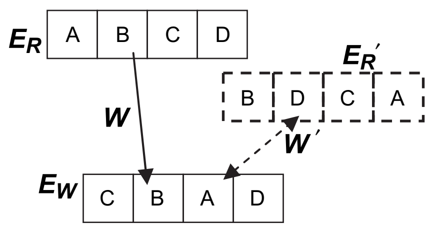

Fig. 4 – Detecting attacks using two reference orders.

E_R and E_R' sequences using W and W′, respectively. If any of the secret orders are violated then at least one of the data entries has been attacked. For example, if E_R was the *Area* and E_R' was the *Lowx*. The output of the above algorithm will be the set of MBRs sorted twice; once sorted according to *Area* and another sorted according to *Lowx*.

Some of the small attacks might not be detected by the *Lowx* reference order but they are detected by the *Area* Reference order and vice versa. Consequently, the overall detection rate will improve. The higher detection rate comes at the extra cost needed to store W′ for each node. Section 4.5.7 shows that if the two reference orders A and B are independent from each other the detection rate of the combined reference order is roughly the summation of the detection rates when any of A or B is used by itself.

4. Security analysis

This section discusses the robustness of the proposed technique, which is measured by the ability to detect malicious alterations. This type of watermarks is called *fragile watermarking*. Watermarking techniques can be classified as *robust* or *fragile*. Robust watermark, which is used mainly for copyright protection, should be able to withstand modification attempts like cropping, compression, transformation, *etc*. On the other hand, fragile watermarks, which are used for data integrity, should be sensitive to modifications. Because of the characteristics of data and types of attacks traditional data integrity methods like simple hash and digital signatures are not suitable for these applications (see Section 4). Unauthorized changes to the R-tree or any of its objects are considered an attack. This includes:

- Deleting one or more data objects or R-tree nodes;
- Inserting one or more data objects or R-tree nodes; or
- Modifying[2] one or more data objects or R-tree nodes.

Note that R-tree nodes are represented by entries in the parent node. Thus, inserting, deleting, or modifying an R-tree node at *level i* is equivalent to inserting, deleting or modifying an entry in *level* $i+1$. Among these types of attacks, the modification attack is the hardest to detect. Insertion attacks would result in changing the number of watermarked entries. As mentioned in Section 2.3, the integrity detection algorithm is very sensitive to the number of entries and thus attacks can be detected easily. This will be always detected unless the replaced object has the exact same size and location.

An attack is considered successful if the attacker altered any of the object attributes e.g., its location or its size and the integrity check algorithm fails to detect the alteration. This can happen in one of two cases:

I. Attacker updated the watermark after altering the data value
II. The change is so small that is not detected by the watermarking detection algorithm

To address the first concern, updating the watermark requires the knowledge of the values of the watermark and the reference order. These two values should be kept secret and known only to authorized users, thus, the attacker would not be able to readjust the order of entries to match the watermark. However, small alteration to data objects might not be detected. If the attack is so small to the extent that it does not violate the secret order of entries, the algorithm fails to detect the attack. This is the second concern in crypto analysis and it is further studied in Section 4.5 using both real and synthetic datasets.

The watermark and reference order are weaved in the structure of the R-tree. Like other watermarking techniques, if the watermark value is compromised or some authorized users lose their privileges the watermark value needs to be changed and thus rebuild the R-tree. The complexity of this operation is $O(n)$.

4.1. *Comparison with simple hash and digital signature*

The proposed technique helps in detecting the integrity of the data stored in R-trees. Traditionally, simple hash and, the more sophisticated, digital signature techniques have been used for checking data integrity.

The simple hash uses a one-to-many function to calculate a digest (or a hash value) that corresponds to the data object or document under consideration. Although hash does not guarantee that the digest is unique, algorithms like MD5 and SHA guarantee no collision with very high probability. Hash can be applied to R-trees in two ways. One way is to apply the hash function to the whole R-tree, and thus, produces one digest. The digest can be used to check whether the whole R-tree (database) has been modified or not. This technique cannot be used to identify and localize changes. In this case the digest would act as a signature and it can be kept secret. Unfortunately this solution is impractical because the digest will change every time insertion, deletion, or update occurs and thus the new digest needs to be communicated to all authorized users. Alternatively, hash can be applied to each node separately. In this case, there will be different digest for each node. Typical R-tree contains thousands of nodes, which mean we need to maintain a prohibitory number of digests and communicate them to authorized users. Note that storing the digest with the corresponding node is not safe as the attacker can alter the node and update the digest accordingly.

Digital signature techniques offer solution to the confidentiality of the digest that the simple hash suffers from. In

[2] Replacing an object or a node is equivalent to modification attack.

the digital signature, the digest is encrypted with the private key of the database owner and thus, attackers cannot change the digest to match the altered data. Similar to simple hashing, applying the digital signature technique on the whole R-tree would result in one signature that will be sensitive to any change, insertion or deletion to any of the R-tree data objects, and thus is not practical. Moreover, the failure of integrity check would mean that the whole R-tree (and the database) is not integral and thus un-usable.

The other possibility is to apply the digital signature algorithm to each R-tree node separately and produce one signature for each node. However, the proposed fragile watermarking technique has two major advantages over using the digital signature:

1. Calculating the digital signature is much more costly than the proposed technique, which consist mainly of few circular shifts. On the other hand, in digital signature one needs to implement two major steps: secure hashing and public key encryption. RSA algorithms (the implementation public key encryption) is computationally expensive (1000–10,000 more expensive than symmetric key cryptography).
2. Digital signature technique produces one signature for each node in the R-tree and thus requires non-trivial extra storage. Moreover, maintaining this large number of signatures and their correspondence with the appropriate nodes is non-trivial. One can argue to store the signature inside the R-tree node. However, this would require changing the R-tree data structure and reduce the fan out (number of children) of the R-tree. Reducing the fan out directly increases query access time.

4.2. *Analytical formula for detection rate*

In this section we derive a formula that gives the probability of detecting malicious attack using the proposed technique. The formula is derived for a specific reference order. In the following derivation the *Lowx* reference order is assumed.

Let the object anchor be the lower left corner of the MBR of an entry and it is marked as ‘‘Black’’ dot in Fig. 5. Assume that anchors are uniformly distributed in the space. The size of MBRs can be of any distribution. Let *x* denote the distance between the anchor of an entry E_i and the anchor of the closet entry along the X-axis. For the simplicity of the derivation and verification of the formula, we assume that the attack is always on the *Lowx* corner.

The node, shown in Fig. 5 (represents the universe in this derivation), has a size *h* along the X-axis and it contains *k* entries. Using the uniformity assumption, the average distance between object anchors is:

$$\bar{x} = \frac{h}{k} \tag{1}$$

Let us assume that the attack value *x* can have any value $0 \rightarrow k$ i.e., *x* is a uniformly distribution over the node space. Thus, the pdf of the attack *x* is:

$$f(x) = \frac{1}{k} \tag{2}$$

Attack will be detected if $x > \bar{x}$; where $\bar{x}$ is the average distance between object anchors.

$$P(\text{attack detection}) = P(x > \bar{x}) = \int_{\bar{x}}^{h} \frac{1}{h} dx = \frac{x}{h} \Big|_{\bar{x}}^{h}$$

$$P(\text{attack detection}) = P(x > \bar{x}) = \frac{h - \bar{x}}{h} = \frac{h - h/k}{h} = \frac{kh - h/k}{h} = \frac{kh - h}{kh} \tag{3}$$

Equation (3) gives the probability of detecting an attack assuming that the attack value is uniformly distributed between 0 and *k*.

4.3. *Victim identification*

When at most one entry is attacked per node, victim entries can be identified with high probability using the proposed watermarking scheme; moreover one can predict the nature of the attack, e.g., increasing or decreasing the victim size. This can be better explained by an example. Let {a, b, c, d, e} be the list of entries in an R-tree node (refer to Fig. 6). Let us further assume that *Area* is the secret reference order used and the sorted list of entries according to *Area* is {a, b, c, d, e}.

Fig. 6A shows the five entries (before attack) sorted according to the areas of their MBRs. Assume that Mallory attacked entry b by increasing its size. The integrity check algorithm uses the watermark value to restore the initial reference order {a, b, c, d, e} and it expects that the areas of the re-arranged entries are monotonically increasing. But because of Mallory unauthorized changes to b, the order was disturbed as shown in Fig. 6B. The resized entry b is supposed to come between entry d and entry e as shown in Fig. 8A. Since it is assumed that only one entry is attacked, one can conclude that b is the victim entry and that its size has been increased.

Note that if the increase in b size is small as shown in Fig. 8C such that b should be located between entry c and entry d according to the order of the size then one would expect that either b or c has been attacked. There are two

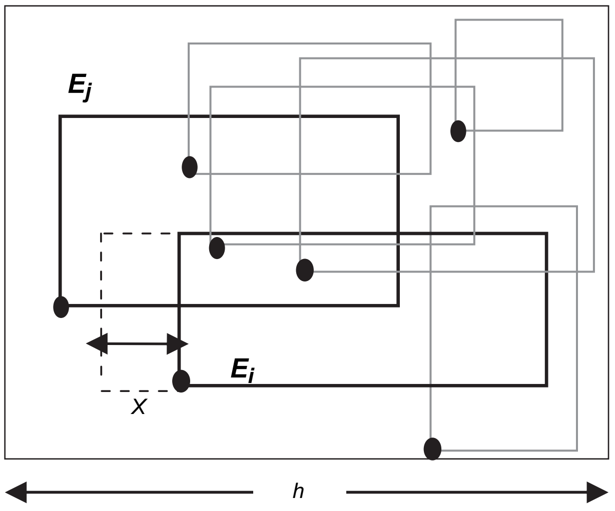

Fig. 5 – An R-tree node. Dotted area is unauthorized alteration.

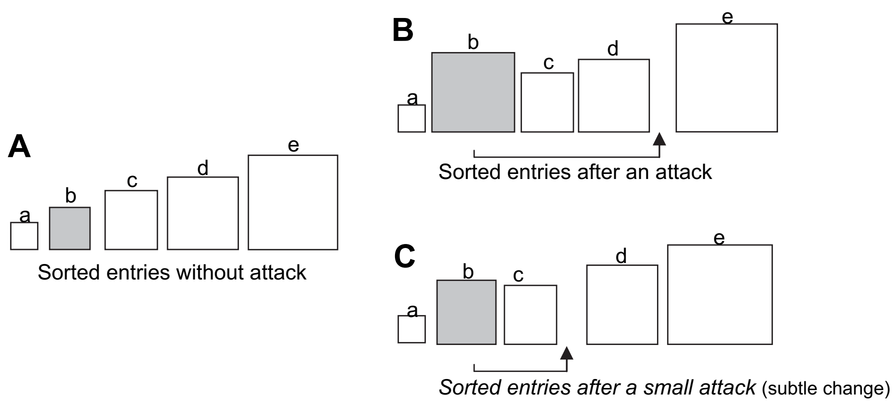
A
a
b
c
d
e
Sorted entries without attack
B
a
b
c
d
e
Sorted entries after an attack
C
a
b
c
d
e
Sorted entries after a small attack (subtle change)

Fig. 6 – Victim identification example using *Area* secret order.

other special cases where victim cannot be identified, when Mallory increases the size of the last entry in the sequence or decreases the size of the first entry in the sequence. Although the above example used *Area* as a reference order, but the idea works equally to other reference order that sorts entries according to one or more features or properties of the objects. Note that if the *hash* is used as reference order, victim node cannot be identified because it is not directly related to entry properties that can be attacked.

4.4. *The effect of reference order* E_R *on the robustness of the watermark*

Section 2.4 lists the different attacks that can be launched against the watermarked R-tree. Note that the attack type depends on the application data and thus some attacks are not applicable for some application data. For example, if the database stores two dimensional regions representing land uses, displacement attack is not applicable in this case.

Since some reference orders are more sensitive than others to a specific attack, the reference order should be selected carefully to match the application and the expected attacks. The *Lowx*, which sorts the node entries on the *x* value of the lower left corner of the corresponding MBR, can only detect attacks on one of the four sides of the MBR. *Area*, which sorts the entries according to the area of the MBR, can catch insertion attacks, deletion attacks and modification attacks. However, *Area* is blind to displacement attack. *Hash* is sensitive to all four attacks mentioned in Section 2.4. However, it cannot be used to identify the victim entry.

4.5. *Experimental evaluation*

In this section we provide performance evaluation for the proposed watermarking algorithm. Our main focus is to measure the detection rate or the probability of detecting unauthorized alteration. We also measure the effect of the watermark on the performance of the R-tree. In our experiments we use real datasets, from www.rtreeportal.org, representing roads, regions and streams in US.

Normally, modification attacks are more difficult to detect than insertion and deletion attacks; this is because modification might go undetected for two reasons: either because the alteration is small or because the sorting criterion is not sensitive to this specific alteration. Our experiments focus mainly on modification attacks. Attacks are simulated as follow: first decide (randomly) whether it will increase or decrease the size of the victim MBR. Then we randomly select one of the four coordinates (for 2-D case) to attack. Finally, select the amount of attack. Obviously, if the attack changes the size of the MBR significantly, it can be detected easily. In the experiments, the attack amount is selected from a uniform distribution that ranges from 0 to 10% of the original size of the MBR. We repeat the above attack scenario for each entry in the R-tree; then calculate the attack detection rate.

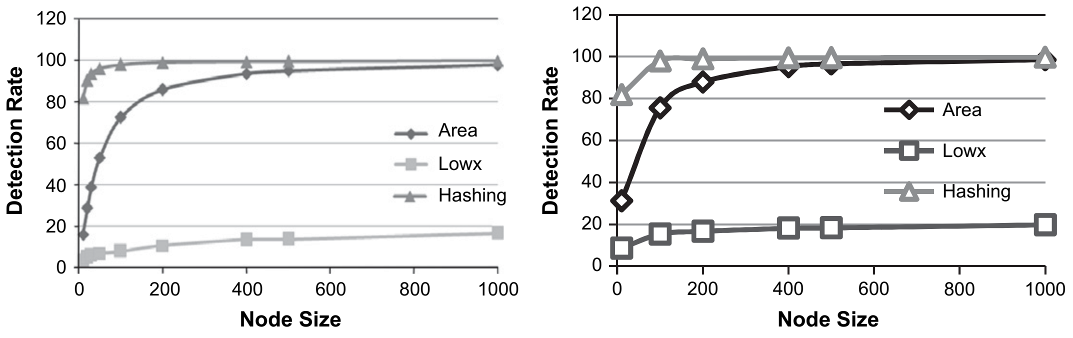

Fig. 7 – Comparing the detection rate of *Lowx*, *Area*, and *Hash*. Left: *Stream* dataset; Right: 4-*States* dataset.

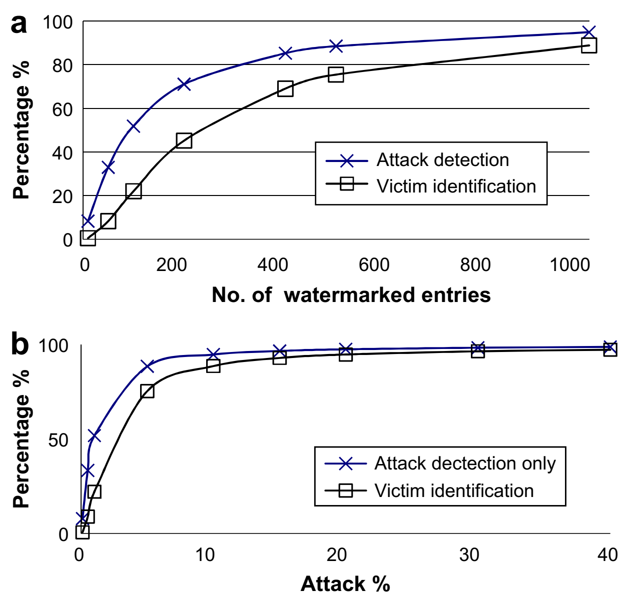

Fig. 8 – Victim identification: a) Attack value is 5% of the original area b) number of watermarked entries is 600.

4.5.1. *Calculating the attack detection rate*

The first experiment in Fig. 7 (left), uses a dataset that consists of 194,000 polylines representing streams of Iowa, Kansas, Missouri and Nebraska (called *Stream* dataset). It measures the detection rate as a function of the node size; it compares the detection rate of three reference orders: *Lowx*, *Area*, and *Hash* (*Perimeter* gives results that are close to *Area* and thus not shown). The detection rate of the *Lowx* is low even for large node sizes. This is because *Lowx* can detect changes in one side of one of the dimensions (x-axis), while it is blind to others. The area is more sensitive than *Lowx* as it catches changes in both dimensions; it achieves detection rate close to 100% when the node sizes are large, while the *Hash* gives the best results as the detection rate reaches close to 100% for node sizes 200 and above. In Fig. 7 (right), we repeated the same experiment for a larger dataset; '4-states' dataset contains 556,696 MBRs representing road network in four states: Iowa, Kansa, Missouri and Nebraska. We noticed also that the detection rate improves as the size of database increases, especially, with small node sizes.

4.5.2. *Victim identification*

Simulation experiments have been carried to measure the success rate in identifying the victim entry and the nature of attack. In the experiment shown in Fig. 8, *Area* reference order is used. The attack is carried by choosing a victim entry at random and changing its size (increasing or decreasing) by a specific percentage of its original size.

The attack detection algorithm is applied to reconstruct the reference order using the secret watermark. The calculated order is tested; if entries do not follow the expected order, then victim identification procedure (as described in Section 4.3) is executed. Note that failure in victim identification means that the algorithm detects an attack but cannot identify which of the two entries is the victim. The attack is repeated many times and the average success rate is calculated. Fig. 8A shows the number of times in which both detection and victim identification are identified as a function of the number of watermarked entries. The second curve in Fig. 8A shows the detection rate only. In this experiment the attack is simulated as 5% increase or decrease in the area of the MBR of the object. The victim identification rate is always lower than the detection rate with up to 20% difference especially when the number of watermarked entries is small.

This difference accounts for the cases where the algorithm detects that there is an attack but cannot identify the victim among the two entries that disturbed the order. Note that the victim identification and nature of the attack identification rate improves significantly as the attack percentage increases (Fig. 8B).

4.5.3. *Displacement attack*

This set of experiments measures the robustness of the proposed watermarking technique in the presence of displacement attacks. The dataset consists of a set of multi-dimensional objects represented by their MBR; the modification attacks can result either in changing the size of the data objects or the location of the objects. Displacement attack is simulated by moving the object in x-direction, or y-direction, or both without changing its size. In each experiment, an object is chosen to be a victim and the movement direction is chosen randomly. The displacement value is proportional to the size of the object. It is chosen to be 10% of the side of the victim MBR. This attack scenario is repeated for each object in the database and the average detection rate is calculated. Fig. 9 shows the detection rate of the displacement attack as

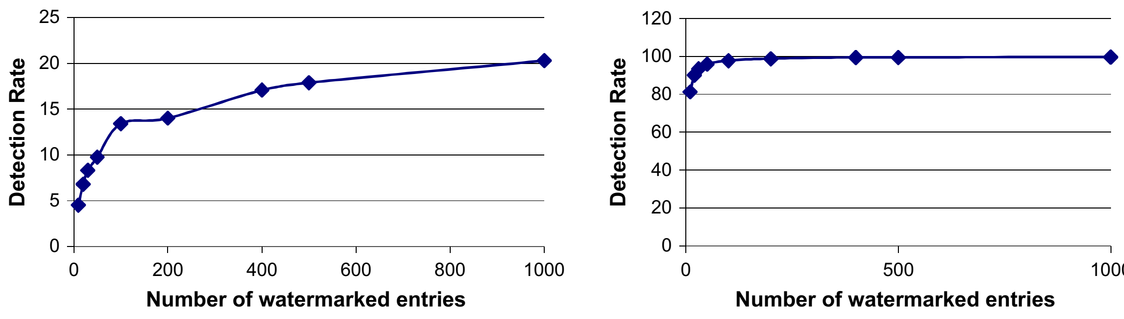

Fig. 9 – Displacement attack. [LEFT] detection rate using *Lowx* reference order; [Right] detection rate using Hash reference order.

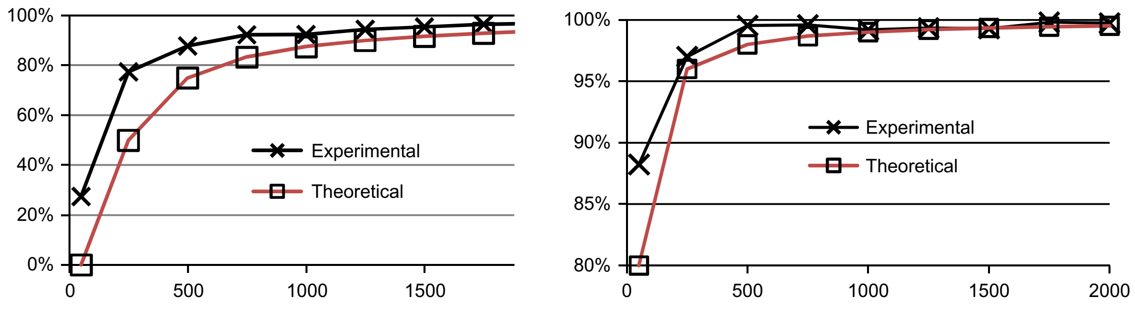

Fig. 10 – Analytical detection rate versus experimental: [Left] attack 1% of the size of the node; [Right] attack accounts to 10% of the size of the node.

a function of the number of watermarked entries. In Fig. 9 [LEFT], the *Lowx* secret order is used, and in Fig. 9 [RIGHT] the *Hash* secret order is used. *Area* secret order gives results close to that of *Hash* secret order and thus is not shown. For *Lowx*, the percentage success is low because *Lowx* secret order can only detect attacks in the x-direction and it is blind to all the attacks in the y-direction. *Hash* secret order detects displacement attacks with rate 99% or more if the number of entries is more than 200.

4.5.4. *Verification of the analytical formula for attack detection*

In this section, the analytical formula for estimating the attack detection success rate is compared to experimental results. Both analytical formulae and experimental results are calculated for the *Lowx* reference order.

Fig. 10 shows the percentage success in attack detection rate versus the number of watermarked entries. The prediction of the attack rate improves with larger attacks. Fig. 10 [Left] shows experiments that simulate the attack by changing the size (increasing or decreasing) by 1% of the node size, while Fig. 10 [Right] uses larger attacks that reach up to 10% of the size of the node.

Each simulation point or setting in the experiment is repeated 100 times and the average detection rate is calculated. The analytical formula predicts the detection rate accurately especially when the number of watermarked entries is large. Because of the statistical nature of the formula, it becomes less accurate when the number of watermarked entries is small.

4.5.5. *Cost of the watermark insertion*

We carried simulation experiments to measure the cost incurred by the watermarking and integrity checking operations. To minimize the operating system overhead, we repeat each experiment 300 times and chose the minimum value. Node size has been changed from 10 to 1000. Table 2 shows that watermark insertion cost in msec as a function of the node size.

Our simulation results showed that the watermark embedding time is increasing with the number of entries in the node. Table 2 compares the insertion time for *Lowx*, *Area*, and *Hash*. The cost of watermark insertion using *Area* and *Lowx* are close to each other; 0.04 ms (for node size 10) and less than 5 ms (for node size 1000). Our experiments also showed that the cost of watermark insertion using *hash* is 20–30 times higher than that for *Area* or *Lowx*. This is because *Hash* uses the expensive MD5 algorithm. However, this cost is incurred only when a node is updated. Node update happens with insertion, deletion or modification operations. This is not affecting the retrieval query operations, which are much more frequent than the update queries. The cost of integrity check is similar to the cost of watermark insertion. Notice that integrity checking can be done in batch processing mode that can be executed off-line.

4.5.6. *The effect of multiple references on the detection rate*

This section shows how multiple watermark–reference pairs, described in Section 3.2, can improve the detection rate of malicious attacks. Using the *Stream* dataset, we employed both *Lowx* and *Lowy* reference orders in watermark insertion. As shown in Fig. 11 the detection rates of *Lowy* are similar to that of *Lowx*; however, the detection rates of the combined *Lowx* and *Lowy* are twice as much the detection rates of any of them. The reason for this high gain is that *Lowx* and *Lowy* are independent parameters and each of them is sensitive to a different set of attacks. *Lowx* is sensitive to attacks at the low side of the x-axis while *Lowy* is sensitive to attacks at the low side of the y-axis.

Table 3 shows the results of another experiment, which uses *Area* and *Hash* reference orders. The combined reference orders still achieve gain in the detection rate but the gain is small, especially with large node sizes. The reason is that *Area* and *Hash* are dependent measures; meaning that some of the attacks that are detected by *Hash* can also be detected by *Area*.

5. Related work

The digital watermarking field was initially nourished by the multimedia research community. A lot of work has been

Table 2 – Watermarking insertion time in msec.

Node Size	*Lowx*	Area	Hash
20	0.04	0.03	0.60
100	0.25	0.02	7.23
200	0.59	0.04	15.00
400	1.37	1.04	31.00
800	3.21	2.55	93.00
1000	4.80	3.40	109.00

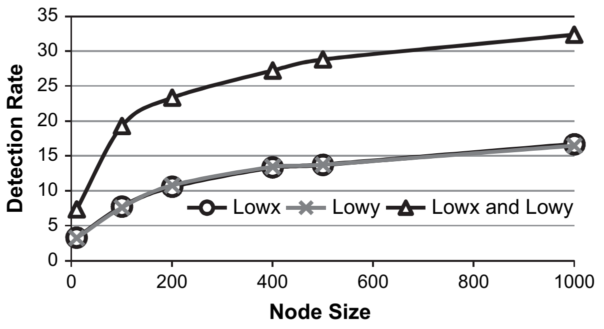

Fig. 11 – Combined reference order *Lowx* and *Lowy*.

proposed for watermarking images, audio, and video objects; (Cox and Miller, 2002) presents a good survey on the recent development of these areas. Recently, the use of watermarking for data integrity and copyright protection has attracted a lot of interest in the research community in areas like databases (Agrawal et al., 2002, 2003; Li et al., 2004; Sion et al., 2003b), software programs (Kamel and Albluwi, 2009; Monden et al., 2000), and XML (Gross-Amblard, 2003).

Traditional techniques in watermarking databases mainly hide a secret message in some of the attributes in the relation. Normally the watermark insertion introduces error to the attribute because it changes its original value; and thus these techniques can be applied only to relations that have attributes that are not sensitive to small errors, e.g., temperature readings (Agrawal et al., 2003). Attributes like salary, prices, and property coordinates cannot be used to host watermarks. The number of watermark-able attributes depends on the nature of the databases. Some relations might have few watermark-able attributes. Other relations might not have attributes that can be watermarked. The detection of malicious alteration is probabilistic and depends on the number of watermark-able attributes. The larger the number of watermark-able attributes is the better the security of the database. Wayner (2002) used a technique similar to our proposed technique for hiding text in the order of a list of songs. Unlike our proposed scheme, this technique is a type of steganography, and thus, it should be as robust as possible to preserve the hidden message even after malicious attacks.

A technique for watermarking databases was introduced in Agrawal et al. (2003) to protect the copyright of the databases. The main idea is to insert the watermark in the least significant bits in the values of some of the attributes in the database. This slightly degrades the data. The authors suggest choosing fields that are not sensitive to small changes, e.g., temperature. Li et al. (2003) extended the previous technique to insert multiple bits so that potential illegal distributors can be tracked. Sion et al. (2003b) presents a robust watermarking scheme that sorts all the tuples and divides them into non-intersecting subsets. A single watermark bit is embedded in each subset by modifying the distribution of tuple values. Watermark bits are embedded more than one time and an error control coding scheme is used to recover the embedded bits. This scheme is not suitable for very dynamic databases with frequent updates, because of the expensive maintenance cost. Guo et al. (2006) proposed a technique for relational database watermarking. The proposed technique arranges the tuples in groups. For each group of tuples, a two dimensional grid of watermarks is created. Along one dimension, a set of watermarks is calculated for selected fields across all tuples. Along the other dimension a watermark is created for each tuple across all the fields. The watermark, which is stored in the least significant bits, is created as a function of the values of the attributes using secure hash function. An attribute change would affect two intersecting watermarks; a horizontal one (at the tuple level) and a vertical one (at the attribute level) that spans all the tuples in the group. Identifying the affected watermarks would identify the altered attribute. This way attacks can be detected and the victim attribute can be identified. Guo et al. (2007) proposed a fragile watermarking for data streaming. Data stream readings are divided into groups. The watermark of every two groups is stored in the first of the two in a chain fashion. The watermark is the hash of all the values in the group and it is stored in the least significant bits of the data readings.

Goodrich et al. (2005) proposed a technique for data forensics of main memory data structures. The technique is based on a new reduced-randomness construction for non-adaptive combinatorial group testing and it hides information in main memory data structures e.g., arrays, linked list, binary search tree and hash tables to enable them to detect any alteration in the data stored. Watermarking XML documents was studied by Gross-Amblard (2003) and Sion et al. (2003a). The idea is to hide the watermark in certain values in a way that preserves the answers to certain queries.

Pang and Tan (2004) present a scheme for checking the integrity of the database query result for edge computing. The proposed scheme is based on Merkle Hash Tree. Each tuple is treated as a leaf node and a verifiable B+− tree. The tree is constructed by adding a signed digest for every attribute and for each leaf node recursively until the root node is reached.

Table 3 – Combined reference order *Area* and *Hash*.

Fanout	Detection Rate		
	Area	Hash	Area + Hash
10	15.9	81.9	84.6
20	28.7	90.5	93.3
30	38.9	93.5	96.1
50	53.0	96.1	98.2
100	72.6	98.0	99.4

6. Conclusion and future work

This paper proposed an effective watermarking scheme for R-trees for detecting malicious attacks, e.g., modification, insertion, displacement, and deletion. This watermarking technique can be used either by itself to protect data or in addition to traditional relational watermarking. Prior techniques hide the watermark by changing the values of some attributes and can be applied only if the database contains attributes that can tolerate some errors. This paper advocates watermarking indexes like B-trees, hash table, R-trees, in addition to the relational table to improve alteration detection. We proposed a novel fragile watermarking for R-trees. The new technique embeds the secret message in the order of entries inside the R-tree node, exploiting the fact that the R-tree does not put condition on the order of entries inside its nodes. Thus, the watermark does not change the data values stored in the R-tree and does not require extra space. The proposed algorithms developed a one-to-one mapping between all possible values that the watermark can take and all possible permutations of the entries. To achieve this we used a factorial-based numbering system.

Our experiments show that the proposed technique detects malicious attacks with high probability that reaches to 99% with large node sizes. We also presented a technique for increasing the detection rate at the expense of the cost of the attack detection.

Future work will focus on the design of watermark for other database indexes, e.g., B-trees and Quad-trees. In this paper, a formula for the detection rate is derived for the *Lowx* reference order. The derivation assumes that data are uniformly distributed. In the future, we plan to derive a closed formula for other reference orders and other data distributions.

Acknowledgments

The authors would like to thank the students Ziad Hatahit and Osama Alkoky for their help in writing parts of the code and carrying experiments.

REFERENCES

Agrawal R, Kieman J, Srikant R, Xu Y. Hippocratic databases. VLDB 2002;.

Agrawal R, Haas PJ, Kiernan J. Watermarking relational data: framework, algorithms and analysis. The VLDB Journal 2003; 12(2):157–69.

Albluwi Q, Kamel Ibrahim. Watermarking essential data structures for copyright Protection. In: The 5th international conference on cryptology ad network security CANS06. Suzhou, Dec. 8–10; 2006.

Beckmann N, Kriegel H-P, Schneider R, Seeger B. The R*-tree: an efficient and robust access method for points and rectangles. SIGMOD 1990.

Cox IJ, Miller ML. Electronic watermarking: the first 50 years. EURASIP Journal on Applied Signal Processing 2002;2002(2):126–32.

Goodrich MT, Atallah MJ, Tamassia R. Indexing information for data forensics. ACNS 2005:206–21.

Gross-Amblard D. Query-preserving watermarking of relational databases and XML documents. In: 22nd ACM SIGMOD-SIGACT-SIGART Symp. on principles of database. June 9–12; 2003, p. 191–201.

Guo H, Li Y, Liua A, Jajodia S. A fragile watermarking scheme for detecting malicious modifications of database relations. Information Sciences 2006;176(10):1350–78.

Guo H, Li Y, Jajodia S. Chaining watermarks for detecting malicious modifications to streaming data. Information Sciences 2007;177:281–98.

Guttman A. R-trees: a dynamic structure for spatial searching. ACM SIGMOD 1984:47–57.

Iyer B, Mehrotra S, Mykletun E, Tsudik G, Wu Y. A framework for efficient storage security in RDBMS. IN: Proc. EDBT; 2004, p. 147–64.

Kamel I, Albluwi Q. A robust software watermarking for copyright protection. Computers & Security 2009;28(6):395–409.

Kamel I, Faloutsos C. On packing R-trees. In: Int. conf. on information and knowledge management (CIKM); Nov 1993.

Kamel I, Faloutsos C. Hilbert, R-tree: an improved r-tree using space filling curve. In: 20th international conference on very large databases VLDB 94. Santiago: Chilep; 1994, p. 500–09.

Li Y, Swarup V, Jajodia S. A robust watermarking scheme for relational data. In: Proc. the 13th workshop on information technology and engineering. December 2003, p. 195–200.

Li, Guo H, Jajodia S. Tamper detection and localization for categorical data using fragile watermarks. In: The 4th ACM workshop on digital rights management, CCS04; October 2004.

Manolopoulos Y, Nanopoulos A, Papadopoulos AN, Theodoridis Y. R-trees: theory and applications. Springer, ISBN 1-85233-977-2; 2005.

Monden A, Iida H, Matusmoto K. A practical method for watermarking java programs. Taipei: Computer Software and Applications; 2000.

Pang H, Tan K. Authenticating query results in edge computing. In: Proc. of the IEEE Int. Conf. on data engineering; March 2004.

Qin Z, Ying Y, Jia-jin LE, Yi-shu LUO. Watermark based copyright protection of outsourced database. In: 10th international database engineering and applications symposium (IDEAS'06) p. 301–8.

Sion R, Atallah MJ, Prabhakar SK. Resilient information hiding for abstract semistructures. In: Int. Workshop on Digital Watermarking (IWDW); 2003a.

Sion R, Atallah M, Prabhakar S. Rights protection for relational data. In: Proc. of ACM SIGMOD; 2003b. p. 98–109.

Smarandache F. Definitions, solved and unsolved problems, conjectures, and theorems in number theory and geometry. Xiquan Publishing House; 2000. p. 29.

Sun J, Tao Yufei, Papadias D, Kollios G. Spatio-temporal join selectivity. Information Systems 2006;31(8):793–813.

Wayner P. Disappearing cryptography. 2nd ed. Morgan Kaufmann; 2002.

Dr. Ibrahim Kamel currently, is an associate professor at the department of Electrical and Computer Engineering at University of Sharjah, UAE. Before that, Dr. Kamel was a Lead scientist at Panasonic research laboratory in Princeton, NJ, USA where he was managing several projects, like multimedia retrieval, smart home, and voice over IP. His research interests include database indexing, multimedia information retrieval, information security, and smart appliances. Dr. Kamel has 21 patents in information storage and retrieval. He published more than seventy papers in international Journals and conferences. He co-chaired several international workshops like the International Workshop on Multimedia Information Retrieval 2000 and the International workshop on Mining Spatial and Temporal Data 2001. Dr. Kamel is serving as an associate editor for several International journals. He also participated in many technical program committees of International conferences. He served in the Industrial advisory committee for the Department of Electrical and Computer Engineering, University of Miami, USA and the Center for Advanced Information Processing (CAIP), Rutgers University, USA. He served in several National Science Foundation (NSF) review committees and panels. Dr. Kamel is a senior member of IEEE.