available at www.sciencedirect.com

ScienceDirect

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

Computers & Security

A robust software watermarking for copyright protection

Ibrahim Kamel[a,*], *Qutaiba Albluwi*[b]

[a]*Department of Computer Engineering Sharjah University, P.O. Box 27272, United Arab Emirates*
[b]*School of Computing, Queens University, Canada*

ARTICLE INFO

Article history:
Received 18 August 2008
Received in revised form
13 January 2009
Accepted 30 January 2009

Keywords:
Information security
R-trees
Watermarking
Multimedia indexes
Spatial data structures
Copyright protection

ABSTRACT

This paper advocates protecting software copyright through hiding watermarks in various data structures used by the code, e.g., B+-trees, R-trees, linked lists, etc. Prior proposals hide the watermarks in dummy data structures, e.g., linked lists and graphs that are created, solely for this reason, during the execution of the hosting software. This makes them vulnerable to *subtractive* attacks, because the attacker can remove the dummy data structures without altering the functionality or the semantic of the software program. We argue that hiding watermarks in one or more data structures that are used by the program would make the watermark more robust because disturbing the watermark would affect the semantic and the functionality of the underlying software. The challenge is that the insertion of the watermark should have a minimal effect on the operations and performance of the data structure.

This paper proposes a novel method for watermarking R-tree data structure and its variants. The proposed watermarking technique does not change the values of the stored data objects. It takes advantage of the redundancy in the order of entries inside the R-tree nodes. Entries are arranged relative to a ‘‘secret’’ initial order, known only to the software owner, using a technique based on a numbering system that uses variable radix with factorial base. The addition of the watermark in the R-tree data structure does not affect the performance nor does it increase the size of the R-tree. The paper provides a detailed security analysis and performance evaluation to show that the embedded watermarks are robust and can withstand various types of attacks.

© 2009 Elsevier Ltd. All rights reserved.

1. Introduction

Software piracy is one of the main threats targeting software development. Recent studies (BSA, 2007) show that 35% of the software programs installed in 2006 are pirated. Watermarking means embedding digital information into original work (Johnson et al., 1998). Watermarks are popular in protecting the copyright of multimedia objects, e.g., images (Petitcolas et al., 1999; Johnson et al., 1998), video (Bloom et al., 1999; Wu and Liu, 2003), and audio (Kirovski and Malvar, 2001; Bassia et al., 2001). In addition to copyright protection, watermarking is also used in data integrity and data confidentiality. Unlike data integrity and confidentiality applications, watermarks for copyright protection applications need to be robust and invisible. Recently, there has been a lot of interest in applying watermarking techniques to protect the copyrights of software codes. Unfortunately, most of the work in this area are trade secrets and are not published.

Technically, most of the software watermarking techniques fall under two categories: *static* watermarking (Davidson and Myhrvold, 1996; Moskowitz and Cooperman, 1996) and *dynamic* watermarking (Collberg et al., 2004; Cousot and

* *Corresponding author.*
E-mail addresses: kamel@sharjah.ac.ae (I. Kamel), qutaiba@cs.queensu.ca (Q. Albluwi).

0167-4048/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2009.01.007

Cousot, 2004). In *static* watermarking the watermark is stored in the source code, either in the data section or in the code section. For example, a watermark can be stored in the values of the constants or in debugging information. On the other hand, *dynamic* watermarking stores the watermark in the program's execution state. Static watermarking techniques are considered more fragile as they can be easily attacked by code optimizers or obfuscators (Collberg and Thomborson, 2002). For example, a static watermark that is saved in data strings can be easily attacked by breaking up all strings into substrings scattered over the executable. Unlike static watermarking, in dynamic watermarking the watermarks are generated during the program execution. In general, dynamic watermarks are more robust than static ones and can withstand more sophisticated attacks (Collberg et al., 2004). This paper focuses on dynamic watermarking. Prior techniques in dynamic watermarking hide the watermark in data structures that are built specially for this purpose during the execution of the program. The fact that the data structure is built independent of the semantic and only to host the watermark makes the watermark susceptible to subtractive (or removal) attacks. The operation and semantic of the host program will not be affected by removing the data structures that hide the watermark.

This paper argues that hiding watermarks in data structures, which are used by the program, would make them more robust, because tampering with these data structures would affect the program correctness and/or performance. Software products usually use a number of both memory-based and disk-based data structures, e.g., binary trees, linked lists, graphs, B-trees, and R-trees. In general, each data structure requires a watermarking technique that is different from the others. The diversity in the watermarking techniques used in one program will definitely increase the robustness of the whole system and thus, decreases the likelihood of successful attacks. The proposed technique has the following desirable features.

- The watermark does not distort/change data object values.
- It does not increase the size of the R-tree.
- The embedded watermark does not interfere with R-tree functionality and operations.
- It is resilient to various types of attacks.

The proposed watermarking technique exploits the redundancy in the order of entries in the R-tree node. It carefully reorders the entries inside the R-trees node, relative to a secret order, in a way that corresponds to the watermark value. Given a watermarked node and knowing the reference order, one can extract the value of the embedded watermark. The rest of the paper is organized as follows. In Section 2, different watermarking and threat models are explained. Section 3 discusses some of the factors that affect the robustness of the proposed watermark. Section 4 describes the proposed watermarking algorithm for R-tree data structures. Section 5 presents a numbering system with a factorial base that is suitable for the proposed watermarking algorithm. A threat analysis of the proposed watermarking scheme, under various possible attacks, is presented in Section 6. To evaluate the effect of the watermark on the performance of the R-tree data structures, Section 6.3 shows relative overhead of the watermark on R-tree operations.

2. Software watermarking model

The goal is to protect the copyright of the software after releasing it. It is assumed that the owner sells the executable and keeps the source code for himself. By hiding a message (watermark), the owner can prove that copies of the released executable belong to him. Software programs consist of two major entities: executable code and data. The executable code resides primarily on disk and it is loaded into the main memory during execution. Watermarks that are hidden in the executable code can be easily attacked by code optimization and obfuscation. Here we propose to hide watermark in the data part and more specifically in the way the data structures are built. Main memory data are either loaded from the disk for processing or temporary data generated during execution and can be attacked during the execution of the program only. While permanent data are residing on the disk, in the form of files, and can be attacked at any time (even if the software is not executed).

A watermark may take different forms, e.g., number, string or graph. Without loss of generality, we assume that a watermark (denoted as W) is a numerical value in the form of a large integer that the software owner selects. This assumption is a reasonable one and does not limit the scope of our design, as there are several transformations to map non-numerical values to numerical forms. Collberg and Thomborson (2002) and Samson (1994) discuss how to choose numerical watermark values in a way to provide legal credibility.

The process of adding the watermark (W) to the software is called *Embedding* and the reverse is called *Extraction*. Embedding the watermark without running the software is considered *static watermarking*, and the opposite scenario is called *dynamic watermarking*.

The proposed technique hides the watermark in the structured data generated by the software code for the following reasons.

1. Data structures are essential parts of the software. Normally, changes to the data structure would affect the functionality of the entire software. Thus, any tampering with the data structure by the attacker (Mallory) would affect the functionality of the software.
2. Embedding the watermark in the source code would normally make the watermark less secure than dynamic watermarking (as discussed in Section 2.1)
3. Using the code data structures would result in a more robust watermark than using non-structured data, like, individual local variables. A watermarking scheme embedded in the non-structured data is more vulnerable to code optimization and attacks from adversaries.
4. Typically, commercial database applications use several data structure; and hiding multiple watermarks in the code using different techniques makes the watermark more robust.

2.1. Static and dynamic watermarks

Static watermarks are stored in parts of the program that are not generated during execution. This includes the executable section that contains instructions (code section) or executable that contains data like headers, strings, and debugging information. The work of Moskowitz and Cooperman (1996) is an example of the static watermark, where the watermark is stored in a digital content (image, video, audio, etc) and then that digital content is stored in the static data section of the program. Other examples can be found in Davidson and Myhrvold (1996) where the order of the basic blocks in the executable, and more specifically the program control flow graph, is mapped in such a way to reflect a watermark.

An example of dynamic watermarking is given in Collberg and Thomborson (2002) and implemented in Palsberg et al. (2000). The watermark is embedded in the running state of the program. The algorithm starts by mapping the watermark, which is a natural number, to a special data structure called Planted Plane Cubic Tree (PPCT). PPCT can be considered as a modified version of binary trees. The pointers between different nodes in the tree are chosen carefully to represent the watermark value. An offline code is used to build up the PPCT tree in the source code. By special addition of some fields, this code is linked to a base class in the program, and when the program is executed the PPCT will be constructed.

2.2. Threat model

Alice is the owner of the software; she published it (or sold it) after adding her secret watermark. She released only the binary code of her software. Mallory wants to use Alice software without paying intellectual property royalties, thus, he tries to get rid of the watermark that Alice inserted to avoid legal actions. Before exploring various types of attacks that Mallory can invoke, let us define the following terms.

2.2.1. Robust watermark

A robust watermark is a watermark that retains its legibility (can be successfully extracted) after undergoing moderate distortive attacks (Collberg et al., 2002). Like other security concepts, robustness is a relative measure. There is no system that is 100% secure or a watermark that is 100% robust. However, a system is considered *secure or robust enough* if

- destroying or removing the watermark would significantly affect the functionality or the performance of the underlying code;
- the *cost* of breaking into the system exceeds the cost of the system or benefit from breaking into it, or;
- the *time* required to break into the system exceeds the life time of the data. For example, if Mallory wants to remove the watermark from the code before a certain deadline (like, submission date, inspection or court date). If removing the watermark would take more than the available time window, then the watermark is considered secure or robust.

2.2.2. Fragile watermark

A watermark is considered fragile if it gets corrupted with minimal change in the embedding media (or cover). Fragile watermark is used in data integrity. This paper focuses on robust watermarking.

2.2.3. Code obfuscation

Code obfuscation is a set of transformations applied to a program that makes it difficult to reverse engineer while preserving the semantic and specifications. Code obfuscation is similar to code optimization, except that, obfuscation maximizes obscurity, whereas, optimization minimizes execution time.

2.2.4. Reverse engineering attack

Mallory would use reverse engineering (through de-compilation), to figure out the functionality of the code or identify the watermark and remove it. Normally this is a time consuming and difficult process, especially if code obfuscation is used. In Collberg et al. (2002) it has been argued that the time needed to reverse engineer an obfuscated code is comparable to the time needed to rewrite the code from scratch and thus reverse engineering is not a viable attack scenario. If reverse engineering is the only attack that an adversary can invoke on the watermark, then, according to the above definition, the watermark is considered *robust enough*.

Attacks against watermarks fall in three main categories (Collberg et al., 2002; Razeen et al., 2004):

1. *Subtractive attacks*. In this type of attack, Mallory tries to remove the watermark, originally inserted by Alice, from the code. However, by doing so he might damage parts of the program or some of its functionalities. The attack is considered successful if the software still retains enough original content to be of value to Mallory.
2. *Distortion attacks*. Mallory might not be able to remove the watermark all together but he might be able to damage or distort the watermark in a way that Alice cannot prove ownership of the code. Mallory should be willing to accept some degradation in the quality or the performance of the stolen software.
3. *Additive attacks*. Mallory can insert his own watermark to the software. The new watermark can either replace the original watermark or is inserted in addition to the original watermark and thus it would be difficult to prove which watermark was inserted first.

3. Design considerations

This section discusses some of the design issues that affect the performance of the proposed watermarking scheme.

3.1. Memory-based data structure versus disk-based data structures

Application data structures can be either memory-based or disk-based. Designing a watermark for memory-based data structure would differ from disk-based data structures in several aspects. First, it is clear that the watermarking technique used for memory-based data structures should be dynamic in the sense that it should be inserted and extracted during the execution of the code. On the other hand, watermarking schemes for disk-based data structures can use both static and dynamic methods. Another difference is the vulnerability to attacks. Memory-based data structures are vulnerable to attacks only while the code is running. However, disk-based data structures can be attacked during the code execution or offline using external codes. This makes the design of robust watermarking schemes for disk-based data structures more challenging. This paper presents a watermarking scheme for one of the most popular disk-based data structures, namely, R-trees.

3.2. *Number of watermark instances*

Every data structure is composed of repeated minor blocks (e.g. nodes) that shape out the overall structure. These blocks might be implemented as nodes; for example; a node in a k-d tree or a node in a B+-tree or in an R-tree. There are three possible scenarios in hiding the watermark: hide the watermark in a single node, hide the watermark in multiple nodes or hide the watermark in all nodes.

Hiding the watermark in a single node or a subset of the nodes might contribute to the secrecy of the watermark, especially if the adversary knows that there is a hidden watermark. In this case the adversary needs to figure out the node (or nodes) at which the watermark is stored to be able to attack them. On the other hand, if the attacker is not aware of the existence of the watermark, the special treatment of specific node by the program would raise the suspicion of the adversary that there might be a watermark in these nodes and thus, would try to attack this node or remove it. It is desirable in this case to hide the watermark in a set of nodes that are critical to the operation of the data structure (like the root), where their absence will highly affect data structure functionality.

Note also that hiding a watermark in one node or subset of nodes would complicate the design of the software because of the high overhead needed to continuously monitor the status of these special nodes. For example, if the only node that contains the watermark is deleted, another node should be chosen to host the watermark. For highly dynamic data structures (e.g. a database with frequent insertions and deletions), this overhead is non-trivial and thus would negatively affect the performance of the software.

Hiding the watermark in all nodes makes subtractive attacks more difficult. The adversary will not be able to delete watermarked nodes to get rid of the watermark. At the same time, it does not require any monitoring of the structure as the watermark will exist in all nodes. However, the disadvantage of watermarking all nodes is the overhead, especially if the watermarking algorithm is costly, and thus, degrading the overall performance of the software. The proposed watermarking scheme hides the watermark in all the nodes in the R-tree. Even though the cost of embedding a watermark is relatively small (as shown in Section 6.3), the proposed scheme uses a *lazy* update strategy in which, the watermark is updated only when a node is created or deleted.

3.3. *Data value versus data structure watermarks*

An efficient watermarking algorithm identifies redundancies in the value or the structure of the object (called cover) which can be exploited in hiding the watermark. For example, the topology (i.e. how different nodes are connected together) might be redundant in some data structures. A watermark can be embedded in the data items themselves by slightly altering their values. Also a watermark can be stored in the data structure which describes relationships between various data items. In both cases, the watermark should have minimal effect on the value of the data and the overall semantic of the data structure.

Watermarking data values are mainly inherited from the classical watermarking schemes used in images, audio, and video objects. Similar ideas are proposed in Agrawal et al. (2002) in the context of relational databases. The authors proposed to hide a watermark in selected fields in the database by modifying some bit values. To thwart possible attacks, bits used to hide the watermark are selected according to a secrete key. The watermark is hidden in fields or attributes that are insensitive to small changes.

Alternatively, the watermark can be hidden in the data structure without the need to change data values. There are several techniques proposed in the literature that mainly hide a watermark in *artificial* structures or graphs (Collberg and Thomborson, 2002; Venkatesan et al., 2001). This can be achieved, for example by, adding one or more nodes or add data entries to the data structure in a way that corresponds to the watermark value. Notice that these new nodes or data entries do not contribute nor interfere with the program semantic. Of course, this contributes to the vulnerability of the method as these new nodes or data entries will be subject to subtractive attacks.

The proposed scheme does not hide the watermark in data values but rather in essential components of the data structure by exploiting the redundancy in the order of entries within a node. The watermark changes the order of the entries in R-tree nodes. The tree can function properly regardless of how the entries are arranged within nodes. The following section gives a brief introduction to the R-tree data structures and their properties.

4. R-tree watermark design

R-trees are extension of the B+-tree for multidimensional objects (Guttman, 1984; Kamel and Faloutsos, 1993, 1994; Beckmann et al., 1990). For simplicity, this discussion uses data in a two-dimensional space; however, an R-tree and its variants work for any number of dimensions. A geometric object is represented by its minimum bounding rectangle (MBR); see Fig. 1. 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 as depicted in Fig. 2. 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-tree can guarantee at least 50% space utilization and at the same time remains balanced.

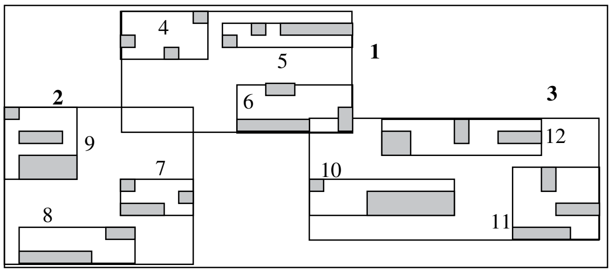

Fig. 1 – Illustrates data rectangles (in black), organized in an R-tree with fanout 3.

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 maximum height of the tree can be calculated using the following formula:

$$h = |\log_m N - 1| \quad (1)$$

Where, N is the number of objects. 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. The split operation is the least frequent operation, but at the same time it is a slow and costly operation. Many techniques have been developed to improve its performance (Beckmann et al., 1990; Kamel and Faloutsos, 1993, 1994). Contrary to the B-tree, R-trees do not put conditions 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; Roussopoulos and Leifker, 1985), Hilbert R-tree (Kamel and Faloutsos, 1994), the R+-tree (Sellis et al., 1987), and the R*-tree (Beckmann et al., 1990).

4.1. *Location of the watermark*

The actual number of entries (fanout) in the R-tree node varies from one node to another depending on the size of the data and the order of insertion. However, R-trees guarantee that nodes are at least half full. To avoid frequent and significant update overhead, the proposed watermarking algorithm uses only the first $M/2$ entries that are guaranteed by R-trees to be non-empty. This way the watermark has minimal overhead on the insertion and deletion operations. A typical R-tree node can have couple of hundred entries, so applying the watermark to half of the maximum number of entries would still allow W to be a large number. Corollary 5.1, presented in Section 5.3, gives a bound on the values of W that can be stored in a given R-tree node.

R-trees have two types of nodes: leaf nodes and non-leaf nodes. However, the structure of the leaf nodes is similar to the structure of the non-leaf nodes.[1] The proposed watermarking scheme does not differentiate between leaf and non-leaf nodes and treat them similarly. Recall, watermark can be stored in one node, multiple nodes or all the nodes.

Contrary to the common practice in image watermarking, hiding the watermark in a single or subset of the R-tree nodes is easier to detect. This is because the code will treat the watermarked node(s) differently, which might alert the adversary that those nodes might be hosting watermarks. Thus, the proposed scheme stores the watermark in all the nodes. One might also hide multiple messages or a large file. This is useful in steganography applications where a large message or a file need to be hidden in an object (e.g., image, audio or video clip, software code, etc.). In such applications, the file can be divided into small messages; each message is embedded in one node in the R-tree. On the other hand, hiding the same watermark (as adopted in this paper) in all nodes offers redundancy that makes the watermarking scheme more robust against possible subtractive attacks.

4.2. *The effect of R-tree operations on the watermark*

Operations that are performed on R-trees can be classified into two main categories: retrieval operations and update operations. Examples of retrieval operations are range queries, nearest neighbor queries and set theoretic queries. Update operations include insertion, split, deletion, and merge. In the following we will investigate the effect of various operations on the watermark.

4.2.1. *Retrieval operations*

These types of operations do not interfere with the watermark because they do not change the data or the structure of the R-tree. However, the watermark can affect the performance and/or the correctness of these operations. Section 6 shows that the proposed watermarking technique does not affect the correctness of R-tree retrieval operations and has negligible effect on the performance (in terms of response time) of the update operations in the R-trees. The update operations affect the data and the structure of the R-tree. The problem arises when these operations change the order of the entries inside the node, because these changes might corrupt the watermark. In this case, a corrective action needs to be taken or the watermark should be reinserted.

4.2.2. *Split operations*

Splits take place after insertions when nodes overflow. The overflowed node is split into two nodes each with at least $M/2$

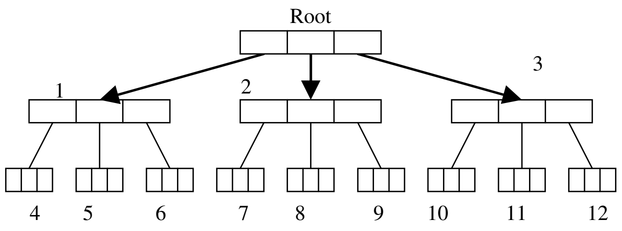

Fig. 2 – Example of an R-tree with fanout 3.

[1] Entries in the leaf nodes point to other R-tree nodes while entries in the leaf nodes point to the object description.

entries. The watermark embedding algorithm (see Section 4.4) should be invoked after a split is performed to rebuild the watermark.

4.2.3. *Insertion operation*

Insertion adds a new entry to an existing R-tree node. Existing entries in the node are stored in a sequence according to the watermark value. Recall that nodes in an R-tree are guaranteed at least half full. The new entry is inserted in the first available location which will be in the second half of the node.[2] But since the proposed algorithm uses only the first $M/2$ entries for watermarking so inserting a new entry (or object) will not affect the existing watermark. Thus, there is no need to invoke the watermarking algorithm after insertion operations.

4.2.4. *Deletion operations*

Deletion operations remove entries from anywhere inside the node. Thus, deletion might affect the existing watermark stored in the node. Moreover, a merge operation might be needed if number of entries fall below the threshold ($M/2$). Thus the watermarking algorithm should be invoked after the deletion and merge operations.

Since the R-tree is stored on the disk, Mallory (attacker) might try to change the order of the entries and corrupt the watermark even if the database application is not running (offline attack). But since the watermarking algorithm is crafted inside the code, Alice can always prove her claim by inserting enough objects to the database to cause at least one node split and create new nodes that carry her watermark. The following theorem gives the number of objects that one needs to insert to guarantee at least one split in each node.

Theorem 1. *For any R-tree of height $h \geq 2$, all leaf nodes are going to split at least once if T entries are added, where T is given by:*

$$T = M^h - m * \left(M^{h-2} + 1\right) + 1$$

Proof 1. Since each non-leaf node can have a maximum of M entries, then maximum number of leaf nodes $= M^{h-1}$. Also since the minimum number of leaf nodes occurs when adding a new level to the tree, such that the new level has number of nodes equal to the maximum number of nodes in the previous level (M^{h-1}) plus the new added node which caused the bottom-up growth of the tree. This is represented by the term $M^{h-2} + 1$

[Example, if $M = 6$ and the current height is 4, then the maximum number of nodes in the 4th level is 6^3 and the minimum is $6^2 + 1$, which is one node extra than the maximum number of nodes of the 3rd level.]

Therefore, the maximum number of data objects (or entries) at the leaf level is given by:

$$E_{\max} = M * \left(M^{h-1}\right) = M^h$$

Minimum number of leaf entries

$$E_{\min} = m\left(M^{h-2} + 1\right)$$

The number of insertions that guarantees at least one split is given by $(E_{\max} - E_{\min} + 1)$. Where

$$E_{\max} - E_{\min} + 1 = M^h - \left[m\left(M^{h-2} + 1\right)\right] + 1 \qquad \square$$

[2] The first $M/2$ entries are inserted by the split algorithm not the insertion algorithm.

4.3. *Watermarking R-tree: the basic idea*

The proposed watermarking technique exploits redundancy in the order of entries within an R-tree node. Entries inside an R-tree node are rearranged in such a way to reflect the watermark W. Later, from the special arrangements of entries we can extract W to prove ownership. To achieve that, we establish a one-to-one mapping between all the permutations of entries within a node at one side and all possible values of W at the other side. The objective of this step is mainly to ensure that each value of W would map to one, and only one, arrangement of entries. This requirement turned out to be non-trivial specially when working with decimal numbering system.

We developed a numbering system based on *factorial numbers*, Variable-base Factorial Numbering System VF-NS (Section 5.2) suitable for this mapping. The proposed numbering system and its advantages over the common decimal system are described in detail in Section 5. The value of the watermark W is converted from its decimal format to the VF-NS format (W_F) for the purpose of producing a unique arrangement of entries. VF-NS uses a numbering system with variable base, meaning, the base of digit i is different from the base of digit j, $\forall\ i \neq j$. The base value of digit number i equal $i!$ (on the contrary, the base of all digits in the binary system is always 2).

First, the embedding algorithm sorts entries in R-tree node according to a reference order E_R. E_R acts as a secret key and is known to the owner only. This step is necessary to be able to extract the watermark. Then W_F is used to shuffle the entries to reflect the watermark. Starting from the reference order E_R, we use left-circular shift operations to change the order of entries. The number of shift operations depends on the value W_F; each node entry E_i moves a number of positions depending on the values of one of the digits of W_F.

This can be better explained by an example. Let us assume that there are four entries A, B, C, and D in the R-tree node and $W_F =$ "311". The reference order E_R is {A,B,C,D}. Since W_F has three digits, then the shuffling operation will be repeated three times, as illustrated in Fig. 3. The first digit in W is 3, then all entries will be shifted to the left three positions, resulting in the new order {D, A, B, C}, Fig. 3(a). In the second round, the first entry {D} will be fixed (not included in the rotation) and the entries {A, B, C} will be shifted only once because the second digit in W_F is "1", resulting in a new intermediate order {D, B, C, A}. Similarly, in the third round entries {D} and {B} will be excluded from the rotation. The remaining two entries {C} and {A} will be shifted once since the third digit in W_F is "1". The final (watermarked) order of the entries would be {D, B, A, C}. We denote the result node as E_W.

The watermark W can be extracted by comparing the watermarked entries to the reference order E_R. We use a procedure similar to that in Fig. 3, where we use circular

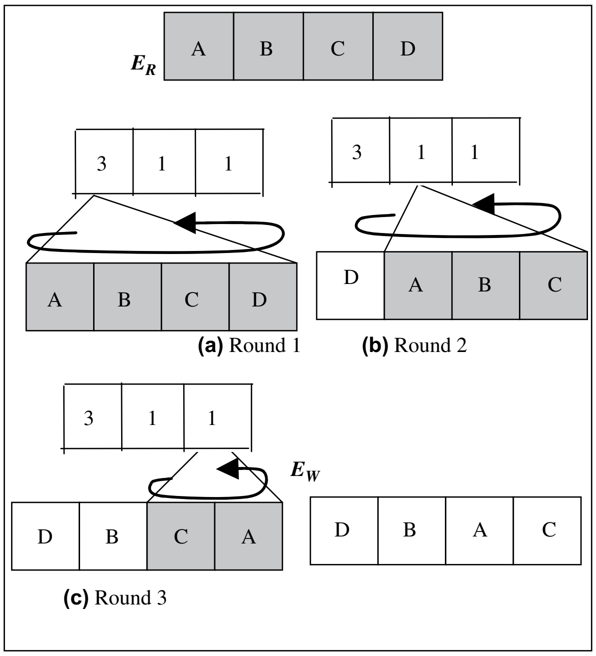

Fig. 3 – Example of watermark embedding.

shifts to align symbols from E_R to E_W. The number of shifts needed to align the first entry (symbol) is the first digit in E_W. W_F can be extracted by comparing various entries in W_F with E_R order.

The following two subsections provide detailed description of the proposed algorithms for embedding and extracting watermarks in R-trees. Without loss of generality, the watermark is assumed to be numerical value.

4.4. *Watermark embedding algorithm*

The problem that is addressed in this section can be formulated as follows.

We need to develop an encoding scheme that takes as an input (1) a numeric value W; (2) a set of entries E; and (3) a reference order of entries E_R. The output of this algorithm is a new order E_W such that:

- E_W contains same set of entries in E and E_R;
- given E_W and E_R, one can uniquely extracts W;
- the algorithms for determining E_W and extracting W are efficient and simple.

The initialization step converts the watermark W that is represented in decimal system to W_F that is represented in VF-NS system (as described in Section 5.2). Recall that the order of entries inside the R-tree node is arbitrary and depends on the order at which data objects are inserted. To be able to retrieve W from the watermarked node, the entries E should be sorted relative to some default order E_R. This order should be secret and known only to Alice (the owner of the database). E_R will be used to extract W later. E_R can be any arbitrary order known to the owner or a sort according to some index. For example entries can be sorted on the x-coordinate of the upper left corner of the Minimum Bounding Rectangle (MBR) (Roussopoulos and Leifker, 1985) or sorted according to the Hilbert order (Kamel and Faloutsos, 1993, 1994).

The shuffling algorithm mainly applies circular-left shift on subsets of entries starting from the order E_R. The number of location shifted equal to $W_F[i]$ where $W_F[i]$ is the value of the ith digit in watermark. Fig. 4 outlines the main steps in the watermark embedding process.

4.5. *Extraction algorithm*

The extraction algorithm is the reverse of the watermarking algorithm. Given a watermarked node and the reference order E_R we need to retrieve the value of W_F and hence W. This is achieved by applying the circular shifts on E_R multiple times in order to make it look like E_W. The number of applied shifts is used to construct the watermark W_F (which can in turn be converted to W). This can be better explained by an example. E contains four entries A, B, C, and D. Let us assume that the default order E_R is {A, B, C, D}. The watermarked order (as it exists in the R-tree node) is E_W = {D, B, A, C}.

The algorithm mainly tries to align each element from E_R with the corresponding element from E_W. The first element in E_W is "D"; therefore, we need to apply three left-circular shifts on E_R in order to achieve this alignment as shown in Fig. 5.

The value "3" is the most significant digit in W_F (in Variable-base Factorial Numbering System as described in Section 5.2). Now E_R = {D, A, B, C}; the entry "D" is in its final position and will not be included in subsequent circular shift operations. Then apply the same procedure to the next entry "B"; shift the entries {A, B, C} one position to the left. Thus, second digit of W_F is 1 and the resulting E_R is {D, B, C, A}. Now "D" and "B" are in their final positions and will not be included in future shift operations. Finally, to align the last two entries in E_R, A, and C should be shifted one position. The final result of W_F would be "311", which can then be converted to the original value W in the decimal format. The extraction algorithm is described in pseudo-code in Fig. 6. The function *Calculate-Distance*() calculates the number of circular shift operations that are needed to align an element from E_R with the corresponding element from E_W. Both embedding and extraction algorithms have been implemented and their performance is shown in Section 6.3.

5. Variable-base factorial number system VF-NS

This section, describes a numbering system, called Variable-base Factorial Numbering System VF-NS that is suitable for the proposed watermarking algorithms. Let us first define the problem formally.

5.1. *Problem description*

Starting from the problem definition in Section 4.3, we want to create a one-to-one mapping between all permutations of R-tree node entries and all possible watermarking values W. This is viewed as dealing with two domains: the permutation

```
Algorithm Name:
Embedding Algorithm

Input:
E is a list of k entries.
In our case k refers to the first M/2 entries.
Watermark $W_F$, number of digits in $W_F$ is k-1

Output: $E_W$

Algorithm:
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 that contains entries from x to y
$W_F$[i] is the i^th digit in W
```

Fig. 4 – Watermark embedding algorithm.

domain P and integer domain R, which contains all possible values of W.

The size of P is a function of the number of entries used in watermarking and equals to the number of permutations. Recall that R-trees guarantee that the first half of the node is always full. Since the watermarking algorithm uses half of the entries in the R-tree node, then the number of permutations is (M/2)! (where M is the node size), or for simplicity $m!$. The following condition needs to be satisfied:

$$|R| \geq m! \qquad (2)$$

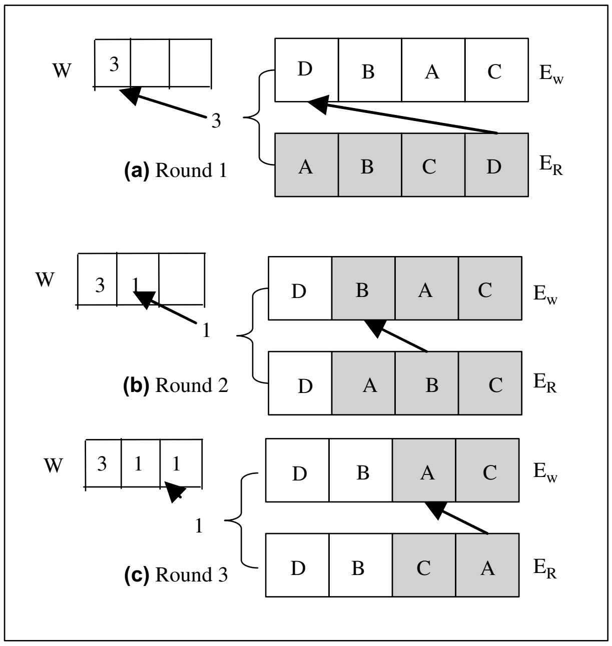

Fig. 5 – Example of extracting W_F from E_R and E_W. Circular shift applied to shaded entries only.

Eq. (2) indicates that the size of the domain R should be greater than or equal to the number of permutations of m entries. After defining the boundaries of the two domains, we need to determine a reversible mapping which guarantees that for each value of W there is one and only one P_i. This condition is needed to be able to extract W to prove authorship.

Without loss of generality we assume that W is an integer value. The domain of integer numbers, R, satisfies Eq. (2) above for some integer value with t digits. Notice that the number of distinct permutations of the entries $E_1, E_2, E_3,\ldots,E_k$ is a function of (k!) while the number of different values of a t digits integer value (represented in the decimal system) is proportional to 10^t (2^t for binary number system). There is no linear relationship between the value of k and the value of t such that $10^t = k!$. Notice also that single increment in the ith digit number 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 elements of P even though the size of $|R| \geq m!$. However, a number system that uses factorial operation as a radix would be more suitable to deal with P than number systems with constant radix, like decimal, binary, octal,…, etc. We propose to represent W in a number system based on the factorial operator. The following subsections give brief description of the proposed numbering system and highlight some of its interesting features that are related to our problem.

5.2. *Description and characteristics of VF-NS number system*

In the factorial system the base of each digit is the factorial of that place index, e.g. place n is of base n!.

Any integer can be represented in the form (see Theorem 2 for the proof):

```
Algorithm Name:
Extraction Algorithm

Input:
Watermarked node N_W
Reference node N_R

Output:
W_F: watermark in the factorial format.

Algorithm:
Node temp = N_R       //copy reference node
int k = N_W.size
int  W_F [ k-1]       //W has k-1 places

for(i = k-1; i > 0; i--){
//find the distance of the entry E_{i+1} between the
// sub-node N-S_i and temp
      W_F[i] = CalculateDistance (N_W-Si; temp; N_W[i + 1])
      temp = left CircularShift(temp, W_F[i])
      temp = temp-S_{i-1}
}
```

Fig. 6 – Algorithm for extracting the watermark W_F.

$$\sum_{k=1}^{n}[a_k * k!] \quad (3)$$

Moreover, the kth digit can take the values from 0 to k only; the least significant digit (1st 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). For example, in the base-10 (decimal) numbering system, each digit can take values from 0 to 9. The integer 859 in decimal (base-10) can be represented as

$$(859)_{10} = (1*1!) + (0*2!) + (3*3!) + (0*4!) + (1*5!) + (1*6!)$$

which is equivalent to 110301 in the factorial system.

$$(859_{10}) = (110301)_{\mathrm{VF-NS}}$$

We found that the proposed number system is similar to an unpopular numbering system dating back to 1800s and is referred to as factorial number system (Smarandache, 2000). Based on the above, we can define the Variable-base Factorial Numbering System (VF-NS) as: *a number 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, as will be proved in Theorem 3.

Eq. (3) can be used to convert from VF-NS to decimal system. Fig. 7 shows an algorithm to convert a decimal number to VF-NS number.

Theorem 2 proves that every integer can be presented in Eq. (3), thus satisfying condition 1, and Theorem 3 proves that each integer is represented in a unique way satisfying the second condition.

Theorem 2. *Let $n \in N$. Then there exist a natural number k such that*

$$n = \sum_{i=1}^{k} a_i i!, \quad 0 \le a_i \le i$$

Proof 2. Clearly the result holds true for $n = 1$. So let us assume that it is true for $n = m$, i.e.,

$$m = \sum_{i=1}^{k} a_i i!, \quad 0 \le a_i \le i$$

for some natural number k and whole numbers $a_1,\ldots,a_k$.

Now, either $a_i = i$ for $i = 1,\ldots,k$ or $a_i < i$ for some $1 \ge i \ge k$.

In the first case,

$$m + 1 = 1 + \sum_{i=1}^{k} i \cdot i! = (k+1)!$$

In the second case, let $1 \ge j \ge k$ be the first index such that $a_j \ge j$. Then

$$m + 1 = (a_j + 1)j! + \sum_{i=j+1}^{k} a_i i!, \quad 0 \le a_i \le i$$

and so by the principle of mathematical induction, the result holds true for all n. □

Theorem 3. *Let $n \in N$. Then there exist unique whole numbers $a_1,\ldots,a_k$ such that*

$$n = \sum_{i=1}^{k} a_i i!, \quad 0 \le a_i \le i$$

```
Algorithm Name:
Decimal to factorial conversion

Input:
A number in decimal format

Output:
Factorial correspondent of the decimal input

Algorithm:
//this algorithm converts from decimal to factorial system
1.//Find number of digits n:
Choose largest L such that: L ! < m
        n = L
        m = int [n]
2.//Calculate the FBV-NS digits:
//assume digits are numbered as
//{ n, n-1, n-2…, 2, 1}
for (i = n; i>1;i--){
    Find largest k in the range [i, 0] such that
           k* ( i !) < x
     m [i] = k
     x = x - (k*(i !))
}
//remainder is put in the least significant digit
   m [i] = x
```

Fig. 7 – Conversion from decimal to factorial.

Proof 3. Suppose $n = \sum_{i=1}^{k} a_i i!$, and $n = \sum_{i=1}^{l} b_i i!$

Claim 1. *$k = l$. If not, say $k < l$, then $b_l \geq 1$ and so*

$$n = \sum_{i=1}^{l} b_i i! \geq 1! \geq (k+1)!$$

But

$$n = \sum_{i=1}^{k} a_i i! \leq \sum_{i=1}^{l} ii! = (k+1)! - 1 < (k+1)!$$

Which is a contradiction. Hence, $k = l$.

Claim 2.

$a_i = b_i$ for all i

First, we show that $a_k = b_k$. If not, say $a_i < b_i$, then

$$n = \sum_{i=1}^{l} b_i i! \geq b_k k! \geq (a_k+1)k!$$

However,

$$n = \sum_{i=1}^{k} a_i i! = \sum_{i=1}^{k-1} a_i i! + a_k k! \leq \sum_{i=1}^{k-1} ii! + a_k k! = k! - 1 + a_k k!$$
$$= (a_k+1)k! - 1 < (a_k+1)k!$$

This is a contradiction. Hence, $a_k = b_k$

But, now, we have

$$\sum_{i=1}^{k-1} ii! = n - a_k k! = \sum_{i=1}^{l} b_i i!$$

Therefore, $a_{k-1} = b_{k-1}$.

Proceeding inductively in a backward fashion, we conclude that $a_i = b_i$ for all i. □

For condition 3, we have shown in the extraction algorithm that we were able to extract the watermark from the permutation order of entries. Therefore, the VF-NS system provides us with a suitable solution for our problem.

Notice that the value of the watermark that can be hidden in R-tree node depends on the node size. The following theorem determines the largest watermark value that can be hidden in an R-tree node that contains m entries.

Theorem 4. *The decimal value that corresponds to the largest m-digits VF-NS number equal to:*

$(m+1)! - 1$

Proof. Let x_d be the decimal value of the largest factorial number that can be stored in m places.

From Eq. (3):

$$x_d = \sum_{k=1}^{n} [a_k * k!] \quad \text{(a)}$$

We know from the definition of factorial numbers that a_k can have values in the range [0, k].

Since we want to find the largest number that can fit in m places, we give a_k the maximum possible value which is k. Eq. (a) can be rewritten as:

$$x_d = \sum_{k=1}^{n} [k \times k!] \tag{b}$$

We want to prove that:

$$\sum_{k=1}^{n} [k \times k!] = (k+1)! - 1 \tag{c}$$

Using induction:

(1) for $k = 1, x = 1 * 1 = 1 = (1+1)! - 1$
(2) assume Eq. (c) is true, we want to prove that Eq. (c) is valid for $k+1$:

$$\sum_{k=1}^{n+1} [k \times k!] = (n+2)! - 1 \tag{d}$$

$$\sum_{k=1}^{n+1} [k * k!] = \sum_{k=1}^{n} k * k + (n+1)(n+1)!$$

From Eq. (c) we substitute $\sum_{k=1}^{n} [k * k!]$ with $(n+1)! - 1$:

$$(n+1)! - 1 + (n+1)(n+1)! = (n+1)!(n+2) - 1 = (n+2)! - 1\square$$

Starting from the above theorem we move to the next section, to show the relationship between the factorial number system and permutation generation.

5.3. *VF-NS number system and permutations*

There is an interesting relationship between the factorial number system and permutations. This comes from the fact that the number of permutations of k items is actually $k!$. The following corollary, derived from Theorem 4, highlights this relationship.

Corollary 5.1. *The permutation of k items equals to the maximum factorial number that can be saved in $k-1$ places.*

Proof. Suppose we have k items.
The permutation of k items $= P_k^k = k!$
Excluding the initial state,

$$k! - 1 \tag{a}$$

Let L be a factorial number saved in $k-1$ places.
Using theorem 4: the maximum value of L is:

$$((k-1)+1)! - 1 = k! - 1 \tag{b}$$

from Eqs. (a) and (b) Corollary is proved.

The above corollary clarifies the relationship between the size of the node and the size of the watermark W that it can hide. For example, if page size is 8 KB, then an R-tree node can store up to 400 entries. Recall that the watermarking algorithm will only use the first half of the entries in the watermarking process.

Thus, if the node maximum capacity $M = 400$, then 200 entries can be used for the watermarking process. Then using Corollary 5.1, each node can hide a numerical value of W up to:

$$200! = 7.89 \times 10^{374}$$

6. Watermark evaluation

A watermark is considered robust if it stands various attacks and distortion attempts. This section presents threat analysis for the proposed watermarking technique and discusses the effect of the watermark addition on the performance of the R-tree.

6.1. *Threat analysis*

From among the three types of attacks that were discussed in Section 2.2, the additive attack is the easiest attack to invoke because it requires zero knowledge about the watermark used in the program. Mallory does not even need to know whether the program is watermarked or not. On the other hand, subtractive attack requires more knowledge about the location of the watermark and how it is stored. Distortion attack is easier than subtractive attack because the adversary does not need to know detailed information about the exact location of the watermark.

6.1.1. *Subtractive attack*

Subtractive attacks as defined in Section 2.2 cannot be invoked against the proposed watermarking technique because the attacker needs to remove half of the entries in the node. This is because the watermark is not occupying physical space in the R-tree data structure; rather, the watermark is encoded in the relative order of the entries inside the R-tree node. In this case, the performance of the code will be affected significantly or it might not work properly. In case the attacker figured out the reference order E_R, she/he can rearrange the entries of the watermarked nodes according to a new invented watermark and thus claims ownership of the code. However, Alice can still generate new nodes that are carrying her original watermark and prove her ownership.

6.1.2. *Distortive attack*

If Mallory suspects that there is a watermark hidden in the order of entries, he can access the R-tree, using external code (offline) and randomly shuffle all entries in the R-tree node. By doing that, the adversary has corrupted the watermarks of the node under consideration. However, the adversary will not be able to block the watermark of newly created nodes when Alice's code is used. So Alice can prove to the judge that she is the owner of the code after inserting, at least, T new objects (as defined in Theorem 1) in the R-tree to build the watermark again. Notice that the insertion cost increases significantly with the height of the tree h (since the cost of insertion is exponential in h). However, this cost is incurred only in case of attack and to prove ownership; and it does not affect the normal operations of the R-tree. Moreover, practically R-trees are shallow, with a typical height of 4–5 levels.

6.1.3. *Additive and inversion attacks*

In additive attacks, Mallory would blindly add his watermark to the software or the R-tree, making it difficult to recognize which watermark is the authentic one. This attack is also known as the ''*false claim of ownership*''. On the other hand, in inversion attacks, the attacker chooses a random permutation as a secret key. In this case, the attacker can claim that the output of the extraction algorithm is his/her watermark.

In practice, no watermarking technique is immune to these types of attacks (BSA, 2007). One possible solution is to register the watermarks with a third party and time stamp them (Craver et al., 1997; Craver and Katzenbeisser, 2001). This way the real owner can show that his/her watermark is the original because it has an earlier timestamp.

6.1.4. *Replacing the whole R-tree*

Since R-tree is a secondary storage index structure, Mallory might create a new R-tree that does not contain Alice's watermark, yet it is similar to Alice's R-tree and can be read by Alice's code. The dynamic nature of the proposed watermark provides the solution. Although removing the current R-tree and embedding another one would remove the watermark from pre-watermarked nodes, it cannot stop the software from watermarking new nodes. If any split or delete operation is invoked, the watermarking algorithm will be executed. So by inserting enough new objects in the new R-tree, Alice watermark will be created again. Theorem 1 gives the number of objects needed to be inserted to force all the nodes in the R-tree to split at least once.

The above analysis is based on a silent (yet common) assumption that the software is protected and cannot be reverse-engineered. Many current software uses obfuscation and other techniques to hinder the process of reverse engineering. However, if Mallory succeeds in obtaining the original code, then applying subtractive and distortive attacks would be much easier. In our algorithm, if Mallory was able to reverse engineer the code then he can replace the watermarking algorithm with a code that shuffles the R-tree nodes randomly causing meaningless extraction of watermark. However, the accuracy of the current reverse engineering tools (McAfee Research), and the size of code would normally make the cost of reverse engineering an obfuscated code more than the cost of building it from scratch (Collberg et al., 2002).

6.2. *The effect of the watermark on the R-tree performance*

The addition of the watermark to the R-tree has minimal effect on its performance. The most frequent and critical operation in the R-tree is the data retrieval, e.g., range query, nearest neighbor query, and join query. R-tree retrieval algorithm does not assume specific order inside the node and it sequentially checks all the entries inside the node against the query object. Since the proposed algorithm does not alter the order or the content of the node and only changes the order of the entries, thus, the watermark will not affect the retrieval operation.

The watermarking algorithm is dynamically called after each split/merge or delete request. In practice, split and delete operations are less frequent than insertion and retrieval operations. Moreover, the watermarking algorithm uses simple operations, like, circular shift; therefore, the execution time is not significant. Both watermark embedding and extraction are $O(n^2)$, where n is the number of watermarked entries. Recall that the maximum number of entries in the node is M and the proposed algorithms watermark $M/2$ entries. Thus, the complexity of the embedding and extraction algorithms is $O(M)$. Section 6.3 presents simulation experiments that show the expected cost of watermark insertion on the operation of the R-tree.

With respect to the space, the watermark does not affect the size of the tree because watermark does not occupy physical space on the disk. It is rather encoded, relative to a secret order that is known only to the owner, in the order of the entries inside the node.

6.3. *Performance results*

This section provides performance results for the proposed watermarking algorithm. The focus is to measure the effect of adding a watermark on the performance of the traditional R-tree operations. We used the R-tree implementation, provided by *Institut für Geoinformatik und Fernerkundung* in the University of Osnabruck (Baer, 2001). The proposed watermarking algorithms are implemented as separate java classes and invoked from within the R-tree implementation. The watermarking code consists of three main modules: VF-NS numbering module, the watermark insertion module, and the watermark extraction module. The first module takes the watermark, in decimal format, as an input and converts it to the VF-NS numbering system. The watermark insertion and extraction modules are direct implementation of the algorithms provided in Figs. 4 and 6. To avoid overflow as a result of the factorial operator when dealing with large watermark value, factorial numbers are represented as an array of integers. Each element in the array corresponds to a digit place in the VF-NS value. The arithmetic operations were implemented using the methods of the java class *BigInteger*. The experiments were applied to node sizes ranging from 10 to 400.

As discussed in Section 4.1 the watermark can be inserted after every update operation (insertion, deletion, change, split, merge), this will guarantee that the watermark is valid at all time. Alternatively, it can be inserted after split and merge operations only, which will guarantee valid watermark for new nodes. In this experiment, the watermarking insertion algorithm is invoked after each split operation. As objects are inserted, R-tree nodes are filled up until a node overflows and the split operation is invoked. The entries of the overflowed node are distributed among two new nodes. The watermark embedding algorithm is applied to the two new nodes. Two performance measures are recorded; the first one measures the watermark insertion cost. The second measures the overhead incurred by the split operation as a result of invoking the watermark insertion algorithm.

The watermark value is chosen as the largest value that can be stored in a node. Meaning, for a node of size n, the watermark value would be the largest value that can be embedded in $n/2$ digit places. To minimize the operating system overhead, we repeat each experiment 300 times and choose the minimum value. Fig. 8 shows the time cost, in

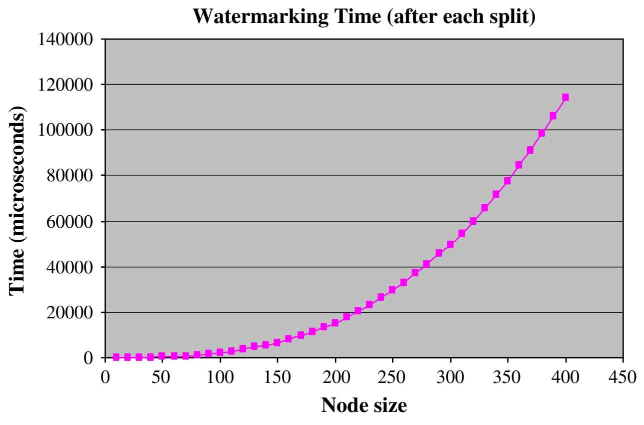

Fig. 8 – Cost of embedding a watermark in an R-tree node.

microseconds, for embedding a watermark in a node of size n, where n varies from 10 to 400 entries.

Fig. 8 shows that the watermark embedding time is increasing quadratically with the number of entries in the node. As R-tree node size increases, the watermark insertion cost can be non-trivial relative to the insertion cost especially with large node sizes. For example, for an R-tree with node size 200, the watermark insertion cost is close to 20 ms. This is equivalent to one disk access. However, this cost is incurred at the split operation only, thus the overhead is acceptable.

In Fig. 9, the y-axis shows the watermark insertion cost relative to the node split cost; the x-axis shows R-tree node fanout (number of entries). Notice that the ratio of the watermark insertion to the split cost is below 0.25 for node sizes below 200; while for node sizes 200–400 this ratio is between 0.25 and 0.5. In practice, most R-trees have node size below 400 and thus the watermark insertion time is always less than half of the split time. We have to keep in mind also that the split operation is the least frequent operation and thus an increase in its time with reasonable fraction (0.2–0.5) can be tolerated. The knee in the second graph is due to the quadratic nature of the split algorithm in the R-tree.

The factorial computation of large integers is generally a costly operation. Since the VF-NS numbering system depends extensively on this computation, we provided an experiment to show that the cost of conversion is small relative to the R-tree operations which are disk based. The

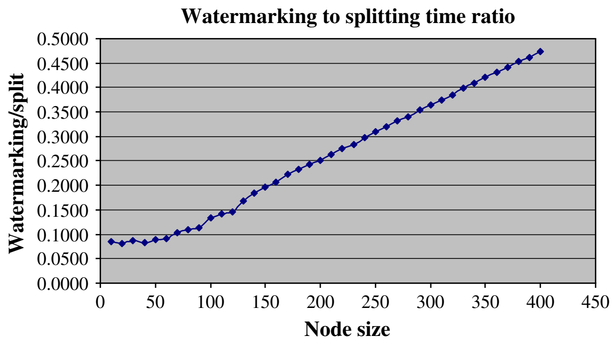

Fig. 9 – The ratio between the embedding time to a node split time.

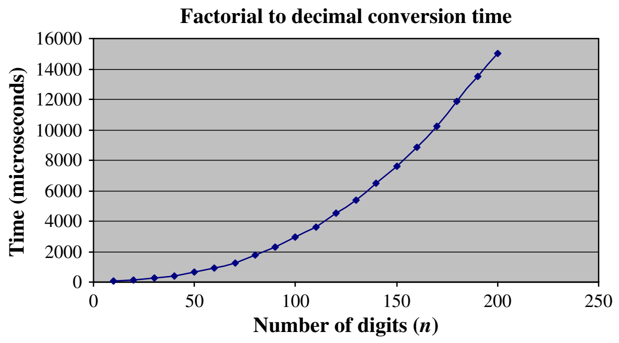

Fig. 10 – Conversion cost from VF-NS to decimal system.

experiment measures the cost of converting the largest VF-NS number that fits in n digits to the decimal system.

For example, the largest VF-NS that can fit in 10 digit places is A987654321 (where A corresponds to the value 10 in decimal). In Fig. 10, the x-axis represents n, the number of digit places in the watermark value; the y-axis represents the time required to convert the watermark from VF-NS to the decimal system. The number of digits n changes from 10 to 200, which corresponds to watermarks that can be used for R-trees of node sizes ranging from 20 to 400. The time is relatively small (milliseconds) and can be neglected compared to the cost of the heavy disk-based operations of the R-tree. Moreover, this conversion is normally applied twice: once when the watermark is inserted and once when the watermark is extracted. Thus, this cost can be always tolerated.

7. Related work

Data security research attracted a lot of interest in the research community in areas like databases (Iyer et al., 2004; Agrawal et al., 2003; Sion et al., 2003b; Scannapieco et al., 2007), data mining (Barbara et al., 2001), and Internet transaction (Bouganim and Pucheral, 2002; Rubin and Greer, 1998).

The first dynamic software watermark was presented in Collberg and Thomborson (1999). The scheme embeds the watermark in the topology of a dummy graph that is built on the heap at runtime. This scheme can be attacked by modifying the pointer topology of the program's fundamental data types, which in turn would change any data structure built at run time. This scheme has been implemented in (Palsberg et al., 2000). The paper provides a detailed discussion on other possible attacks. Khanna and Zane (2000) describe a scheme for encoding information in the weights of a graph representing a map so as to preserve shortest paths. A similar technique was presented in Venkatesan et al. (2001), where a watermark is embedded in a control-flow graph that is never executed. Likewise, Monden et al. (2000) suggested embedding the watermark in a dummy method that is never executed.

A different approach is presented in Cousot and Cousot (2004). The watermark is embedded in the values of some designated local variables during the program execution. The main advantage of this scheme is that the watermark can be recovered even if only small part of the code is available. This scheme has been analyzed in Collberg et al. (2004). The scheme can be attacked by obfuscating the program such that local variables representing the watermark cannot be located.

A technique for watermarking databases was introduced in Agrawal et al. (2003) which slightly degrades the data. The main idea is to insert the watermark in the least significant bits of the data values. The authors suggest choosing fields that are not sensitive to small changes, e.g., temperature. 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. Zhao et al. (2004) proposed a method for protecting the copyright of relational database, but it requires the original copy (un-watermarked) of the database to extract the watermark.

Watermarking XML documents was studied by Gross-Amblard (2003) and Sion et al. (2003a,b). The idea is to hide the watermark in certain values in a way that preserve the answers to certain queries.

8. Conclusions and future work

Data structures make an essential part of the program logic and their removal would manifestly affect the functionality of the program, in this paper we argue that hiding the watermark inside data structures that are used by the software makes the watermark more robust.

The paper presents a novel technique for hiding watermarks in R-tree data structure by reordering entries inside the node. The proposed algorithm takes advantage of the fact that R-trees do not put conditions on the order of entries inside the node. The entries are carefully re-ordered according to the value of the watermark. The new order is calculated relative to a secret order E_R (secret key) using a VF-NS, numbering system with variable base equals to the factorial of the index of the digit. Detailed algorithms for embedding and extraction watermarks are provided. Algorithms that provide a one-to-one correspondence between a numerical value of the watermark W and the permutation of a set of entries are also presented. The watermark does not increase the size of the R-tree nor does it distort the values of data objects. Moreover, the paper provides a threat analysis for our proposed method.

The proposed technique can be applied to other data structures that do not put restriction on the order of the data; however, it is not suitable for data structures that are sensitive to the order of the data. In the future, we will study hiding watermarks in other disk-based data structures like, B+-trees, Quad-trees, k-d tree, etc. in addition to memory-resident data structures like stacks, linked lists, arrays, binary trees, etc.

Acknowledgments

The authors would like to acknowledge Prof. Ragheb Abuseris, from the mathematics division in the University of Sharjah for providing the proofs of theorems 2 and 3 and for reviewing the mathematical derivations in the paper. We also thank the students Ahmad Hatahit and Ayman Agamy for their help in writing parts of the code.

REFERENCES

Agrawal R, Kieman J, Srikant R, Xu Y. Hippocratic databases. In: Proceedings of very large databases (VLDB), 2002, Hong Kong, China.

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

Baer W. R-tree implementation, http://deegree.sourceforge.net/deegree1.x.x_javadoc/org/deegree_impl/io/rtree/RTree.html.

Barbara D, Couto J, Jajodia S. ADAM: a testabed for exploring the use of data mining in intrusion detection. SIGMOD Record 2001;30(4):15–24.

Bassia P, Pitas I, Pitas N. Robust audio watermarking in the time domain. Multimedia IEEE Transaction June 2001;3:232–41.

Beckmann N, Kriegel H-P, Schneider R, Seeger B. The R*-tree: an efficient and robust access method for points and rectangles. ACM SIGMOD, May 1990, p. 322–31.

Bloom JA, Cox IJ, Kalker T, Linnartz J-PM, Miller ML, Traw CB. Copy protection for DVD video. Proceedings of the IEEE July 1999;87(7):1267–76.

Bouganim L, Pucheral P. Chip-secured data access: confidential data on untrusted servers. In: Proceedings of very large databases (VLDB), 2002, Hong Kong China.

Business Software Alliances (BSA). Piracy study, Fourth Annual BSA And IDC global software, http://www.bsa.org/globalstudy/; 2007.

Collberg C, Thomborson, C. Software watermarking: models and dynamic Embeddings. ACM POPL'99, January 1999, p. 311–24.

Collberg C, Thomborson C. Watermarking, tamper-proofing, and obfuscation – tools for software protection. IEEE Transactions on Software Engineering August 2002;28(No. 8).

Collberg C, Nagra J, Thomborson C. A functional taxonomy for software watermarking. In: Proceedings of 25th Australian Computer Science Conference, vol. 4; 2002. 177–86.

Collberg C, Carter E, Debray S, Huntwork A, Linn C, Stepp M. Dynamic path-based software watermarking. Proceedings of the ACM SIGPALN '04 Conference on Programming Language Design and Implementation June 2004;39(6).

Cousot P, Cousot R. An abstract interpretation-based framework for software watermarking. In: Proceedings of the 31st ACM SIGPLAN-SIGACT, 2004, p. 173–85.

Craver S, Katzenbeisser S. Security analysis of public-key watermarking schemes. Proceedings of SPIE Mathematics of Data/Image Coding, Compression and Encryption VI 2001;vol. 4475:172–82.

Craver S, Memon N, Boon-Lock Yeo, Yeung MM. On the invertibility of invisible watermarking techniques. In: Proceedings of the '97 International Conference on Image Processing, vol. 1; 1997. 540 p.

Davidson IR, Myhrvold N. Method and system for generating and auditing a signature for a computer program. Assignee: Microsoft Corporation.; September 1996. US Patent 5559884.

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

Gross-Amblard D. Query-preserving watermarking of relational databases and XML documents. ACM symposium on principles of database systems (PODS), 2003, p. 191–201.

Guttman A. R-trees: a dynamic structure for spatial searching. Proceedings of ACM SIGMOD, June 1984, p. 47–57.

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

Johnson NF, Jajodia S. Exploring steganography: Seeing the unseen. IEEE Computer February 1998;31(No. 2):26–34.

Kamel I, Faloutsos C. On packing R-trees. Second international conference on information and knowledge management (CIKM), November 1993, p. 490–99.

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, 1994, Santiago, Chile, p. 500–9.

Khanna S, Zane F. Watermarking maps: hiding information in structured data. ACM/SIAM symposium on discrete algorithms, 2000, p. 596–605.

Kirovski D, Malvar H. Robust spread-spectrum audio watermarking. In: Proceedings of 2001 IEEE International Conference on Acoustics, Speech and Signal Processing, vol. 3; May 2001. 1345–8.

McAfee Research. State-of-the-art in decompilation and disassembly (SADD), http://www.isso.sparta.com/research/documents/sadd.pdf.

Monden A, Iida H, Matusmoto K. A practical method for watermarking java programs. In: 24th computer software and applications conference (COMPASAC2000), Taipei, October, 2000.

Moskowitz SA, Cooperman M. Method for stega-cipher protection of computer code. Assignee: The Dice Company; January 1996. US Patent 5,745,569.

Palsberg J, Krishnaswamy S, Kwon M, Ma D, Shao Q, Zhang Y. Experience with software watermarking. In: Proceedings of ACSAC'00, 16th annual computer security applications conference, 2000, p. 308–16.

Petitcolas FA, Anderson RJ, Kuhn MG. Information hiding: a survey. Proceedings of the IEEE July 1999;87(7):1062–78. Special issue on protection of multimedia content.

Razeen M, Ali A, Sheikh N. State-of-the-art in software watermarking. Second international workshop on frontiers of information technology, December 20–21, 2004, Islamabad, Pakistan.

Roussopoulos N, Leifker D. Direct spatial search on pictorial databases using packed R-trees. In: Proceedings of ACM SIGMOD, May 1985.

Rubin A, Greer D. A survey of the world wide web security. IEEE Computer September 1998;31(9):34–41.

Samson PR. Apparatus and method for serializing and validating copies of computer software. Assignee: Autodesk, Inc.; February 1994. US Patent 5,287,408.

Scannapieco M, Figotin I, Bertino E, Elmagarmid A. Privacy preserving schema and data matching. SIGMOD Conference 2007, p. 653–64.

Sellis T, Roussopoulos N, Faloutsos C. The R+ tree: a dynamic index for multi-dimensional objects. In: Proceedings of 13th international conference on VLDB, September 1987, p. 507–18.

Sion R, Atallah MJ, Prabhakar SK. Resilient information hiding for abstract semistructures. In: Proceedings of the workshop on digital watermarking (IWDW), 2003a, Seoul, Korea.

Sion R, Atallah MJ, Prabhakar SK. Rights protection for relational data. In: ACM International Conference on Management Data (SIGMOD). ACM; 2003b. p. 98–109.

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

Venkatesan R, Vazirani V, Sinha S. A graph theoretic approach to software watermarking''. Proceedings of the fourth international workshop on information hiding 2001. Lecture Notes In Computer Science April, 2001;2137:157–68.

Wu M, Liu B. Data hiding in image and video: part I – fundamental issues and solutions. IEEE Transactions on Image Processing June 2003;12(No. 6):685.

Zhao Y, Niu XM, Zhao DN. A method of protecting relational databases copyright with cloud watermark. International Journal of Information Technology 2004;1(1):206–10. ISSN: 1305-2403.

Dr. Ibrahim Kamel received Ph.D. degree in Computer Science from University of Maryland, College Park, USA in 1994, and B.Sc. in Computer Engineering from University of Alexandria, Egypt in 1984. Currently, Dr. Kamel is an associate professor at the Department of Electrical and Computer Engineering at University of Sharjah, UAE. He also holds an adjunct professor position at Concordia University, Canada.

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 US patents in information storage and retrieval. He published more than 70 papers in International Journals and Conferences. He co-chaired several international conferences and workshops in information retrieval and databases. Dr. Kamel is serving as an associate editor for several international journals. He also participated in many technical program committees. He served in several National Science Foundation (NSF) review and panel committees. Dr. Kamel is a Senior Member of the IEEE.

Qutaiba Albluwi received his B.Sc. degree from the Department of Electrical and Computer Engineering at University of Sharjah, UAE in 2004 and M.Sc. from Queens University, Canada in 2009. Currently, he is a Ph.D. student at Queens University, Canada.