Watermarking Essential Data Structures for Copyright Protection

Qutaiba Albluwi and Ibrahim Kamel

Department of Computer Engineering
University of Sharjah, UAE
qutaiba@sharjah.ac.ae, kamel@sharjah.ac.ae

Abstract. Software watermarking is a new research area that aims at providing copyright protection for commercial software. It minimizes software piracy by hiding copyright signatures inside the program code or its runtime state. Prior proposals hide the watermarks in dummy data structures, e.g., linked lists and graphs that are created during the execution of the hosting software for this reason. This makes it vulnerable to subtractive attacks, because the attacker can remove the data structure without altering the operation or the semantic of the software program. In this regard, we argue that hiding watermarks in one or more data structures that are used by the program would make the watermark more robust because removing the watermark would alter the semantic and the operations of the underlying software. However, 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 scheme takes advantage of the redundancy in the way the entries within R-tree nodes are ordered. R-trees do not require ordering the entries in a specific way. Node entries are re-ordered in a way to map the watermark. The new order is calculated relative to a "secret" initial order, known only to the software owner, using a technique based on a numbering system that uses variable radix and factorial base. The addition of the watermark in the R-tree data structure neither affects the performance nor increases the size of the R-tree. The paper provides a threat model and analysis to show that the watermarked R-trees are robust and can withstand various types of attacks.

Keywords: Software watermarking, copyright protection, data hiding, indexing, multimedia database.

1 Introduction

Software piracy is one of the main threats targeting software development. A recent study [18] shows that 36% of the software programs used nowadays are pirated. Watermarking is a technique used to provide copyright protection for intellectual properties. Recently there have been a lot of interests in securing and protecting databases [32] [4] [21] [35].

D. Pointcheval, Y. Mu, and K. Chen (Eds.): CANS 2006, LNCS 4301 , pp. 241 – 258, 2006.

Watermarking means the embedding of digital information into the original work [27]. Watermarks are commonly known in copyright protection of multimedia objects, e.g., images [16] [14], video [20] [23], and audio [11],[29]. In addition to copyright protection, watermarking is used in authentication and privacy protection. Unlike authentication applications, watermarks in copyright protection and privacy applications need to be robust and invisible.

Recently, there has been a lot of interest in applying watermarking techniques to protect the copyrights of the data and the software. 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 [17] [39] and *dynamic* watermarking [6][30]. 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 [6]. 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. Usually the watermark is a large integer number[1]. To make the watermark more credible legally, the large integer is chosen to be the multiplication of two large prime numbers [6] [31]. In general, dynamic watermarks are more robust than static ones and can withstand more sophisticated attacks [7]. Our proposed technique falls under the dynamic watermarking category.

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 specifically to house the watermark and it is independent of the application semantic 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.

We argue 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.

Notice that disk-based data structures are easier to attack than memory-based data structures because an adversary does not need to execute the application program to attack the watermark. Rather, the adversary can invoke an off-line attack with an independent code. With our proposed algorithm the owner can prove his/her claim by inserting enough records into the data structure to rebuild the watermark.

[1] This does not limit the scope of these techniques because any text watermark can be mapped to a numeric watermark.

In this paper, we propose a technique for embedding and extracting watermarks in R-tree and its variants. Our proposed technique has the following desirable features:

- Does not interfere with R-tree functionality and operations
- The watermark does not increase the size of the R-tree
- Robust: it is resilient to various types of attacks
- Simple to implement

The proposed watermarking technique makes use of the redundancy in the order of entries in the R-tree node. It carefully reorders the entries inside the R-tree node to map the watermark.

The next section gives a brief description of the R-tree data structure. Section 3 describes the threat model. In section 0 we discuss some design parameters related to the location and the size of the watermark. Section 0 describes our proposed watermarking scheme and outlines the embedding and extraction algorithms. In section 0, we provide a threat analysis and evaluation of the proposed techniques. Finally, the related works and the conclusions are presented in sections 0 and 0 respectively.

2 R-Tree

The original R-tree [1] was first suggested by Guttman; it is the extension of the B-tree for multidimensional objects. For simplicity, we describe R-tree data structure in the context of the 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-tree can guarantee at least 50% space utilization and at the same time remain balanced.

Figure 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 data tree; meaning all leaves appear on the same level. The maximum height of the tree can be calculated using the following formula:

$$h = \left|\log_m N\right| - 1 \tag{1}$$

Where, N is the number of objects.

Search, insert, delete, split and merge are the popular R-tree operations. The most frequent operation is search. When a node overflows, the split routine is invoked to split the node into two. 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 [1]. 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 [28] [15], the R+-tree [41], and the R*-tree [26].

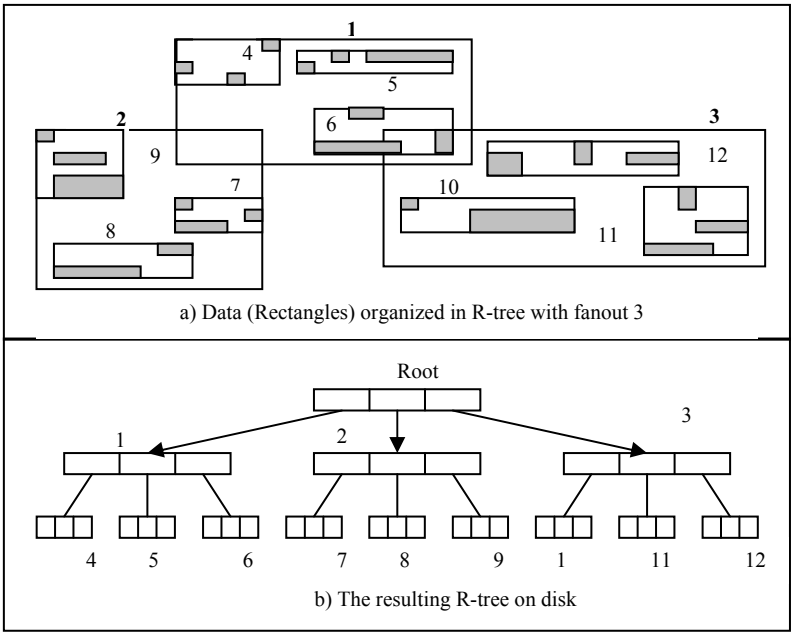

Fig. 1. Example of an R-tree with fan out 3

3 Threat Model

Alice is the owner of the software. She added a watermark to the R-tree. She released only the binary source of her software. Mallory wants to use Alice software without paying intellectual property royalty, 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:

Robust watermark: Robustness and security are relative measures. 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 the *cost* of breaking into the system exceeds the cost of the system or benefit from breaking into it, or
- if the *time* required to break into the system exceeds the life time of the data. Let us assume that 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.

Code obfuscation: is a set of transformations on a program that preserve the same semantic and specification while making it difficult to reverse-engineer the code. Code obfuscation is very similar to code optimization, except that, obfuscation maximizes obscurity, whereas, optimization minimizes execution time

Reverse engineering: Mallory would use reverse engineering (through de-compilation), to figure out the functionality of the code or identify the watermark and remove it. In [5] it has been argued that the time needed to reverse engineer an obfuscated code is similar 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 [8][22]:

1) ***Subtractive attacks:*** In this type of attack, Mallory tries to remove the watermark from the software program inserted by Alice. However, by doing that he might damage parts of the program or some of its functionalities. Thus, 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 can not prove the ownership of the code. Mallory should be willing to accept some degradation in 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 replaces 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.

4 Design Considerations

Throughout the paper, the notation in Table 1 will be used.

Table 1. Notations

W	Watermark
M	Minimum number of entries per node
M	Maximum number of entries per node
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 reference order.
E_W	The set of entries E after embedding the watermark.

The proposed watermarking technique makes use of the fact that R-trees do not put requirements on the order of entries inside the R-tree node and hides the watermark *W* in the order of entries. Thus, the size of watermark (*W)* that can be hidden depends on the size of the node (number of entries).

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 hide the watermark. This way the watermark has minimal overhead on the insertion and deletion operations. Typical R-tree node can have couple of hundred entries, so *W* can be a very large number. Corollary 2.1 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 node[2]. The proposed watermarking algorithms do not differentiate between leaf and non-leaf nodes and treat them similarly.

There are three possible scenarios for selecting the location of the watermark.

- Store the watermark in a single node
- Store carbon copies of the watermark in a selected subset of nodes or
- Store carbon copies of the watermark in all the nodes.

Contrary to the common practice in image watermarking, hiding the watermark in a single or subset of nodes is easier to detect. Since the code will treat the watermarked node(s) differently, this might alert the adversary that those nodes might be hosting watermarks. Thus, the proposed scheme stores the watermark in all the nodes.

Typical R-tree consists of thousands of nodes. Since we are using all the nodes, one might also hide multiple messages or a large file. This is useful for steganography 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 (or message) in all nodes offers redundancy that makes the watermarking scheme more robust against possible subtractive attacks. In this paper, we store the same watermark *W* in all nodes.

4.2 The Effect of R-Tree Operations on the Watermark

The 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. Since the watermark is hidden in the order of entries inside the node, we will investigate the effect of these operations on the entries order.

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

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. On the other hand, 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.

Split Operations: Splits take place after insertions when nodes overflow. The overflowed node is split into two nodes each with at least *M/2* entries. The watermarking algorithm should be invoked after split is performed to rebuild the watermark.

Insertion Operation: Insertion adds a new entry to an existing R-tree node. Entries are stored in a sequence. The new entry is inserted in the first available location which will be in the second half of the node[3]. 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, the watermarking algorithm needs not to be invoked after insertion operations.

Deletion Operations: deletion operations remove entries from anywhere inside the node. Thus, deletion might affect the existing watermark. 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 might try to change the order of the entries and corrupt the watermark even if the program is not running. 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 splits and create new nodes that carry the 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, all leaf nodes are going to split at least once if T entries are added, where T is given by:

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

Proof 1

Since each non-leaf node can have a maximum of *M* entries, then maximum number of leaf nodes = M^{h-1}. Similarly, the minimum number of leaf nodes equal to $M^{h-2} + 1$

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

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

Minimum number of leaf entries

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

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

$$E_{max} - E_{min} + 1 = M^h - [m (M^{h-2} + 1)] + 1$$ □

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

5 Watermarking Algorithm

This section describes the proposed watermarking technique: the watermark embedding algorithm and the watermark extraction algorithm.

5.1 The Basic Idea

The proposed watermarking technique mainly rearranges the entries of the node[4] in such a way to reflect the watermark *W* that we would like to hide. To achieve that we establish a one to one mapping between all the permutations of the entries at one side and all possible values of *W* at the other side. Then we use circular shift operator to move entries to their new positions. After fixing the position of the first entry, we apply the circular shift operator on other entries to fix the position of the second entry and so on and so forth.

Basically, 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 value for *W*. The first problem we faced is that the number of possible values of W represented in decimal radix format does not match the number of permutation *k!*. Moreover, we realized the need to use a flexible numbering system that can work with variable radix and an alphabet with variable size. Popular numbering systems use fixed radix. For example, decimal systems use radix 10, while binary system uses radix 2.

We worked out the requirement and details of a numbering system that fulfills the previous two requirements. Later, we found an unpopular numbering system that was dated back to 1800s that is similar to ours, called factorial numbering system [13] [12]. Before applying the embedding algorithm we convert *W* from the decimal system to a factorial system. The following subsection gives a brief description of the factorial number system, and some of its interesting features.

The embedding algorithm first sorts the entries according to the reference order E_R. Then, the entries are shuffled in such a way to reflect the factorial form of *W*. Starting from the reference order E_R we use circular (left) shift operations to change the order of entries. The number of shift operations depends on the value *W*. Each node entry E_i moves a number of positions depending on the values of one of the digits of *W*.

This can be better explained by an example. Let us assume that there are four entries A, B, C, D in the R-tree node and *W* = "311". The reference order E_R is {A,B,C,D}. Since *W* has three digits, then the shuffling function will go through three rounds, as illustrated in *Fig.2*.

Since the first digit in *W* is 3, during the first round, all entries will be shifted to the left three positions as in *Fig.2.a* a resulting in the new order {D,A,B,C}. In the second round, the entries {A, B, C} will be shifted only once, resulting in {D,B,C,A}. The third round shifts the entries {C, A} one time. The final order of the entries would be {D,B,A,C}.

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.2*. We use circular shift 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 *W*. The extraction algorithm is described in Section 0.

[4] Calculated relative to some presumed reference sequence E_R of entries.

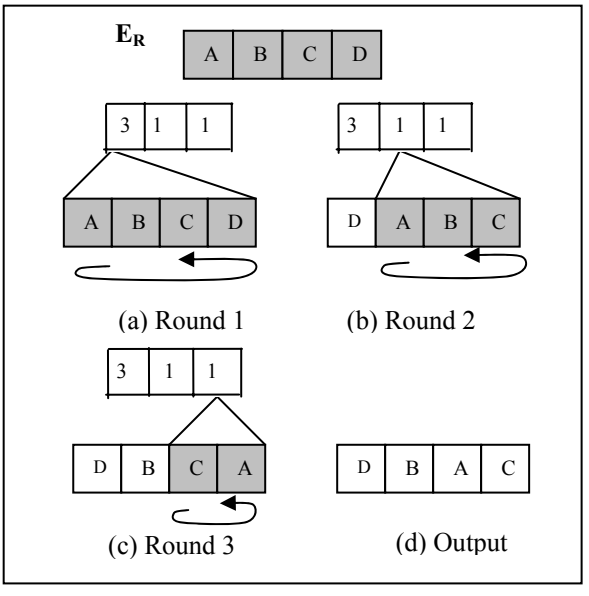

Fig. 2. Example of watermarking embedding

5.2 Factorial 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:

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

For example, the integer 859 can be represented as
(1*1!) + (0*2!) + (3*3!) + (0*4!) + (1*5!) + (1*6!) which is equivalent to 110301 in the factorial system.

Based on the above, we can define a factorial number (FN) as: *a number where the base of the k^{th} place is k!, and the allowed coefficients are between 0 and k.* This condition is important to guarantee one-to-one correspondence between the permutation of the entries and the value of *W*. For example, a four digits number in the factorial system would have the base and coefficient values presented in Table 3.

Table 2. The relationship between the base and the allowed coefficients in the factorial number system.

Place	Base	Coefficient value
1	1! = 1	0,1
2	2! = 2	0,1,2
3	3! = 6	0,1,2,3
4	4! = 24	0,1,2,3,4

The following is an algorithm to convert a number from decimal radix system to factorial radix system:

```
This algorithm converts from decimal to factorial system
Input:
   Decimal number (x)
Output:
   Factorial number (m)
Algorithm:
   1. //Find number of digits n:
   Choose largest L such that: L ! < x
     n = L
     m = int [n]
   2.//Find 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. 3. Conversion from decimal to factorial

To convert from factorial to decimal we use Equation 2. The following theorem and corollary give the largest integer value of *W* that can be embedded in *m* entries.

Theorem 2

The decimal value of the largest m-digits factorial number equal to:

$$(m+1)! - 1 \tag{4}$$

Proof 2

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

From equation (3): $x_d = \sum_{k=1}^{n} [a_k * k!]$ …. (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*. Equation (a) can be rewritten as:

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

We want to prove that :

$$\sum_{k=1}^{n} [k * k!] = (k+1)! - 1 \quad \text{… (c)}$$

Using induction:

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

$$\sum_{k=1}^{n+1}[k*k!] = (n+2)!-1 \quad \ldots \text{(d)}$$

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

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

$$\begin{aligned}(n+1)! - 1 + (n+1)(n+1)! \\ = \quad (n+1)!(n+2) \\ = \quad (n+2)! - 1\end{aligned}$$ □

Corollary 2.1
The permutation of k items equals to the maximum factorial number that can be saved in k-1 places

Proof 2.1
Suppose we have *k* items.
The permutation of k items = P_k^k = k!
Excluding the initial state, k! -1 ….. (a)
Let *L* be a factorial number saved in *k-1* places.
Using theorem 2: the maximum value of *L* is:
((k-1) + 1) ! – 1 = k! – 2 …. (b)
from (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 KBytes, 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. Then if the node maximum capacity *M = 400*, then 200 entries can be used for the watermarking process. Then using Corollary 1, each node can hide a numerical value of *W* up to:

$$200! = 7.89 * 10^{374}.$$

5.3 Embedding Algorithm

The problem that is addressed in this section can be formulated as follow:
Develop an encoding scheme that takes as an input

1) a numeric value W;
2) a set of entries E; and
3) Reference order E_R.

The encoding scheme should produce 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 calculation of E_W and the extraction of W are efficient and simple

The initialization step converts *W* from decimal radix system to the factorial radix system. Recall that the order of entries inside the R-tree node is arbitrary and depends on the order of their arrival. 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) to be able to extract *W* later. E_R can be an arbitrary order 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) 28] or sorted according to the Hilbert order [15].

The shuffling algorithm mainly applies circular-left shift on subsets of entries starting with the order E_R. The number of location shifted equal to *W[i]* where *W[i]* is the value of the i^{th} digit in factorial radix system. The following is a description of the algorithm.

```
Input:
  E with k entries ,
  Watermark W, number of digits in W is k-1
Output:
  Watermarked set (Ew)
Algorithm:
  For ( i = k-1 ; i > 0 ; i--)
   //Circular-left-shift the subset E [x , y] by the value W [i]
         leftCircularShift (W [i] , E[i+1, k ])
  Where: E[x , y] is a subset of E that contains entries from x to y
  and W[i] is the ith digit in W
```

Fig. 4. Shuffling function

5.4 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*. This is achieved by applying the circular shifts on E_R multiple times in order to make it looks like E_W. The number of applied shifts is used to construct the watermark *W*. This can be better explained by a counter example. Let us assume *E* contains four entries A,B,C, and D. The watermarked order is E_W = {D, B, A, C}. Let us also assume that the default order E_R is {A,B,C,D}.

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 3 left-circular shifts on E_R in order to achieve this alignment as shown in Figure 5. The value "3" is the most significant digit in *W* (in factorial radix system). Now E_R = {D,A,B,C}. The entry "D" is in its final position and will not be included in the 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* is 1

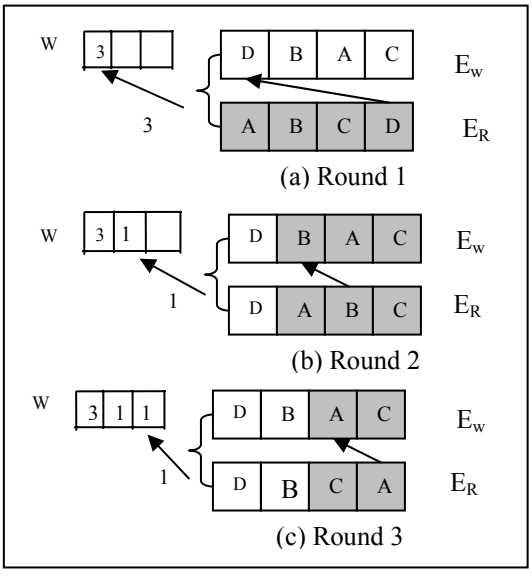

Fig. 5. Example of extracting W from E_R and E_W

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 would be "311".

The extraction algorithm is described in pseudo-code in Figure 5.

```
   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 [ 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 subnode N-S_i and temp
        W [i] = CalculateDistance (N_W -S_i ; temp ; N_W [ i + 1] )
        temp = left CircularShift ( temp , W [i] )
        temp = temp-S_i-1
   }
```

Fig. 6. Extraction Algorithm

In Fig.5 the function *CalculateDistance*() calculates the number of circular shift operations that needed to align an element from E_R with the corresponding element from E_W.

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 3*, 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, the 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 need to know much accurate information about the exact location of the watermark.

Subtractive attack: Subtractive attacks as defined in *Section 3* can not be invoked against the proposed watermarking technique because the watermark is not occupying physical space in the R-tree data structure; rather, it is encoded in the relative order of the entries inside the R-tree node. However, the adversary might try to invoke subtractive attack by deleting the whole R-tree. In this case, the performance of the code will be affected significantly or it might not work properly. Since R-tree is a secondary storage index structure, Mallory might create a new R-tree that does not contain (Alice) watermark, yet it is similar to Alice R-tree and uses it with 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 can not stop the software from watermarking new nodes. If any split or delete operation is invoked, the watermarking algorithm would 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.

Distortion attack: If Mallory suspects that there is a watermark hidden in the order of entries, he can access the R-tree, using external code and randomly shuffle all entries in the R-tree node. Again by doing that, the adversary has only corrupted the current watermarks; however, he will not be able to block future watermarks. Therefore, Alice can prove to the judge that she is the owner of the code after inserting T new objects (as defined in Theorem 1) in the R-tree to build the watermark again.

Additive attack: Here 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 problem is called the "false claim of ownership". In practice, no watermarking technique is immune to this type of attack 6. One possible solution is to register the watermarks with a third party and time stamp them [30] [38] [37]. In this way the real owner can show that his/her watermark is the original because it has older timestamp.

The above analysis is based on the common assumption that the software is protected and can not be reverse-engineered. Many current software uses obfuscation and other techniques to render the process of reverse-engineering. However, if

Mallory succeeds in obtaining the original code, then applying subtractive and distortion attacks would be much easier. In our algorithm, if Mallory was able to reverse engineer 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 [24], 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 [5].

6.2 Effect 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, join query, etc. Neither the watermark itself nor the watermark embedding algorithms affects R-tree retrieval operations.

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 data retrieval operations. Moreover, the watermarking algorithm is a memory-based and it uses simple operations, like, circular shift; therefore, the execution time is negligible.

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

7 Related Work

Security research attracted a lot of interest in the research community in areas like databases [35] [32] [4], data mining [9], and Internet transaction [21][3].

The first dynamic software watermark was presented in [6]. 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 [19]. The paper provides a detailed discussion on the possible attacks. Khanna and Zane [40] describe a scheme for encoding information in the weights of a graph representing a map so as to preserve shortest paths.

Similar technique was presented in [36], where a watermark is embedded in a control-flow graph that is never executed. Likewise, [2] suggested embedding the watermark in a dummy method that is never executed.

A different approach is presented in [30]. The watermark is embedded in values assigned to designated integer local variables during program execution. The main feature of this scheme is that the watermark can be recovered even if only small part of the code is present. This scheme has been analyzed in [7]. The scheme can be attacked by obfuscating the program such that local variables representing the watermark cannot be located.

Database watermarking technique was introduced in [33] which slightly degrades the data. The main idea is to insert the watermark in the least significant bits in the data values. The authors suggest choosing data that are not sensitive to small changes, e.g., temperature. [25] 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 [10] and Sion et al. [34]. 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

In this paper we presented a novel technique for hiding watermarks in R-tree data structure. We utilized the fact that R-trees do not put requirements 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 using a numbering system with variable radix and factorial base. We presented detailed algorithms for embedding and extraction watermarks and an algorithm that provides a one to one correspondence between a numerical value of the watermark (W) and the permutation of a set of entries. Moreover, the watermark does not increase the size of the R-tree. We provided a threat model for software watermarking and a threat analysis for our proposed method.

Since data structures are essential part of the program logic and their removal would affect manifestly the functionality of the program, we argue that hiding the watermark inside data structures that are used by the software makes the watermark more robust. In the future, we will extend our technique to hide watermarks in other data structures like, B-trees, Binary trees, K-D tree etc.

References

1. A. Guttman, “R-trees: A dynamic structure for spatial searching”, Proc ACM SIGMOD, June 1984, pp. 47-57.
2. A. Monden, H. Iida, and K. Matusmoto, “A practical method for watermarking java programs,” *24th computer software and applications conference* (COMPASAC2000), Taipei, Oct, 2000.
3. A. Rubin and D. Greer. “A survey of the world wide web security,” *IEEE Computer 31 (9)*, September 1998, pp. 34-41.
4. B. Iyer, S. Mehrotra, E. Mykletun, G. Tsudik and Y. Wu. “A framework for Efficient Storage Security in RDBMS”. Proc. EDBT 2004, pp. 147-164.
5. C. Collberg and C. Thomborson. “Watermarking, tamper-proofing, and obfuscation- tools for software protection,” *IEEE transactions on software engineering,* vol.28, no. 8, 2002.
6. C. Collberg and C. Thomborson., “Software watermarking: models and dynamic Embeddings,” *ACM POPL`99*, January 1999, pp.311-324.

7. C. Collberg, E. Carter, S. Debray, A. Huntwork, C. Linn, and M. Stepp, "Dynamic path-based software watermarking," *Proceedings of the ACM SIGPALN '04 conference on programming language design and implementation*, vol. 39, issue 6, June 2004.
8. C. Collberg, J. Nagra, and C. Thomborson, "A functional taxonomy for software watermarking," *Proc. 25th Australian Computer Science Conference 2002,* vol 4, pp. 177-186.
9. D. Barbara, J. Couto and S. Jajodia. " ADAM: A testabed for exploring the use of data mining in intrusion detection,". SIGMOD Record, 30 (4), 2001 pp. 15-24.
10. D. Gross-Amblard. "Query-preserving watermarking of relational databases and XML documents," *ACM Symp on Principles of Database Systems (PODS),* 2003, pp. 191-201.
11. D. Kirovski, and H. Malvar, "Robust spread-spectrum audio watermarking," *Proceedings of 2001 IEEE International Conference on Acous tics, Speech and Signal Processing,* vol. 3, May 2001, pp. 1345-1348.
12. D.E. Knuth, "The art of computer programming: Volume 2: seminumerical algorithms," *Addison-Wesley,* Third edition, 1998.
13. F. Smarandache, "Definitions, solved and unsolved problems, conjectures, and theorems in number theory and geometry," Edited by: M.L.Perez. *Xiquan publishing house*, 2000, p. 29.
14. F.A. Petitcolas, R.J. Anderson, and M.G. Kuhn, "Information hiding: a survey," *Proceedings of the IEEE, special issue on protection of multimedia content*, vol. 87, issue 7, July 1999, pp.1062-1078.
15. I. Kamel and C. Faloutsos. "On packing R-trees," *Second Int. Conf. on Information and Knowledge Management (CIKM),* November 1993, pp. 490-499.
16. I.J. Cox and M.L. Miller, "Electronic watermarking: the first 50 years," *EURASIP Journal on Applied Signal Processing,* vol. 2002, issue 2, February 2002, pp. 126-132.
17. I.R. Davidson and N. Myhrvold. "Method and system for generating and auditing a signature for a computer program," *US Patent 5559884*, September 1996, Assignee: Microsoft Corporation.
18. Industrial Design and Construction (IDC) and Business Software Alliance (BSA), "Piracy study," July 2004. http://www.bsaa.com.au/downloads/ PiracyStudy070704.pdf
19. J. Palsberg, S. Krishnaswamy, M. Kwon, D. Ma, Q. Shao, and Y. Zhang. "Experience with software watermarking," *Proceedings of ACSAC'00, 16th Annual Computer security applications conference,* pp. 308-316, 2000.
20. J.A. Bloom, I.J. Cox, T. Kalker, J-P.M. Linnartz, M.L. Miller, and C.B. Traw, "Copy protection for DVD video," *Proceedings of the IEEE* , vol 87, Issue 7, July 1999, pp. 1267 -1276.
21. L. Bouganim and P.Pucheral. "Chip-Secured Data Access: Confidential Data on Untrusted Servers", *Proc. Very Large Databases (VLDB), 2002*, Hong Kong China.
22. M. Razeen, A. Ali and N. Muhammad Sheikh. "*State-of-the-art in software watermarking*", *2nd International Workshop on Frontiers of Information Technology*, December 20-21, 2004, Islamabad, Pakistan
23. M. Wu and B. Liu. "Data hiding in image and video: part I- Fundamental issues and solutions," *IEEE Transactions on image processing*, vol. 12, no. 6, June 2003, p. 685.
24. McAfee Research. "State-of-the-art in decompilation and disassembly (SADD)," http://isso.sparta.com/research/documents/sadd.pdf
25. Michael T. Goodrich, Mikhail J. Atallah and Roberto Tamassia. "Indexing Information for Data Forensics," *ACNS 2005*. pp. 206-221.
26. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. "The R*-tree: an efficient and robust access method for points and rectangles," *ACM SIGMOD*, May 1990, pp. 322-331.

27. Neil F. Johnson and Sushil Jajodia, “Exploring Steganography: Seeing the unseen,” *IEEE Computer*, vol. 31, No. 2, February 1998, pp 26—34.
28. N. Roussopoulos and D. Leifker. “Direct spatial search on pictorial databases using packed R-trees”, *Proc. ACM SIGMOD*, May 1985.
29. P. Bassia, I. Pitas, and N. Nikolaidis, “Robust audio watermarking in the time domain,” *Multimedia IEEE transaction*, vol. 3, June 2001, pp. 232-241.
30. P. Cousot and R. Cousot. “An abstract interpretation-based framework for software watermarking”, *Proceedings of the 31st ACM SIGPLAN-SIGACT,* 2004, pp. 173,185.
31. P.R. Samson. “Apparatus and method for serializing and validating copies of computer software,” *US Patent 5,287,408,* February 1994. Assignee: Autodesk, Inc.
32. R. Agrawal, J. Kieman, R. Srikant and Y.Xu. "Hippocratic Dtabases". *Proc of Very Large Databases (VLDB)*. Hong Kong, China, 2002.
33. R. Agrawal, P.J. Haas and J.Kiernan. “Watermarking relational data: framework, algorithms and analysis” The VLDB journal. vol. 12, issue 2, August 2003, pp. 157-169.
34. R. Sion, M. J. Atallah and S. K Prabhakar. “Resilient information hiding for abstract semistructures,”. *In Proc. Of the Workshop on Digital Watermarking (IWDW),* 2003, Seoul.
35. R. Sion, M. J. Atallah and S. K. Prabhakar. “Rights protection for relational data,” *ACM International Conference on Management Data (SIGMOD)*, 2003, pp. 98-109.
36. R. Venkatesan, V. Vazirani and S. Sinha, “A graph theoretic approach to software watermarking”. *Information Hiding Workshop ’01*, vol 2137, April, 2001, pp. 157-168,
37. S. Craver and S. Katzenbeisser. “Security analysis of public-key watermarking schemes,” *Proceedings of SPIE Mathematics of Data/Image Coding, Compression and Encryption VI*, vol. 4475, 2001, pp. 172-182.
38. S. Craver, N.Memon, B-L. Yeo, and M.Yeung, “On the invertibility of invisible watermarking techniques,” *Proceedings of the ’97 Int. Conf. on Image Processing*. vol. 1, 1997, p.540
39. S.A. Moskowitz and M. Cooperman, “Method for stega-cipher protection of computer code,” *US Patent 5,745,569,* January 1996. Assignee: The Dice Company.
40. S. Khanna and F. Zane. “Watermarking maps: Hiding information in structured data,” *ACM/SIAM Symp on Discrete Algorithms*, 2000, pp. 596-605.
41. T. Sellis, N. Roussopoulos, and C. Faloutsos. “The R+ tree: a dynamic index for multi-dimensional objects,” *Proc. 13th Innternational Conference on VLDB*, September 1987, pp. 507-518