Towards a provably resilient scheme for graph-based watermarking*

Lucila Maria Souza Bento[1,2]
Davidson Boccardo[2]
Raphael Carlos Santos Machado[1,2]
Vinícius Gusmão Pereira de Sá[1]
Jayme Luiz Szwarcfiter[1,2,3]

[1] Instituto de Matemática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil,
lucilabento@ppgi.ufrj.br, vigusmao@dcc.ufrj.br

[2] Instituto Nacional de Metrologia, Qualidade e Tecnologia, Rio de Janeiro, Brasil,
drboccardo@inmetro.gov.br, rcmachado@inmetro.gov.br

[3] COPPE-Sistemas, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil,
jayme@nce.ufrj.br

Abstract

Techniques of watermarking/fingerprinting concern the embedding of identification data into a digital object, allowing for later claims of authorship/ownership and therefore discouraging piracy. Graph-based watermarking schemes comprise an encoding algorithm, which translates a given number (the identifier, usually a positive integer) onto some appropriately tailored graph (the watermark), and a decoding algorithm, which extracts the original identifier from a given watermark. Collberg, Kobourov, Carter and Thomborson (*Error-correcting graphs for software watermarking*, WG'03) introduced one such scheme, meant for software watermarking, in which an integer key was encoded onto a reducible permutation graph. A number of interesting ideas have further improved the original scheme, including the formulation of a particularly promising linear-time codec by Chroni and Nikolopoulos. We extend the work of these authors in various aspects. First, we characterize the class of graphs constituting the image of Chroni and Nikolopoulos's encoding function. Furthermore, we formulate a novel, linear-time decoding algorithm which detects and recovers from ill-intentioned removals of $k \leq 2$ edges. Finally, our results also include the detection of $k \leq 5$ edge modifications (insertions/deletions) in polynomial time and a proof that such bound is tight, so the resilience of the considered watermarking scheme is fully determined. Our proof that graphs of a well-characterized class can detect and recover from bounded-magnitude distortive attacks reinforces the interest in regarding those graphs as possible watermarking solutions to numerous applications.

Keywords. algorithms; reducible permutation graphs; graph-based watermarking; software security

*Work partially supported by Eletrobrás — Distribuição Rondônia, DR/069/2012.

1 Introduction

Preventing the theft of intellectual property has always been a highly sought after goal. In particular, the criminal reproduction of software known as software piracy has become a big concern in recent years. Watermarking/fingerprinting an object is the act of embedding an identifier of authorship/ownership within that object, so to discourage illegal copying.

Different approaches to software watermarking have been devised to date, still none of them was ever proved to be sufficiently resilient, let alone immune, to the numerous forms of program transformation attacks. Naturally, a lot of research towards strengthening such methods has been endeavored. The pioneering graph-based watermarking algorithm was formulated by Davidson and Myrhvold [10]. It inspired the publication, by Venkatesan, Vazirani and Sinha [14], of the first watermarking algorithm where a positive integer—the *key*—was encoded as a special digraph that could be disguised into the control flow graph (CFG) of a program. Other graph-based watermarking schemes include [6, 8, 9, 13].

In this paper, we consider the ingenious graph-based watermarking scheme introduced by Collberg, Kobourov, Carter and Thomborson [7], and afterwards developed and improved upon by Chroni and Nikolopoulos [1].[1] These latter authors proposed a linear-time *codec* (an encoding/decoding procedure) to obtain watermarks that are particular instances of the reducible permutation graphs introduced in [7]. Chroni and Nikolopoulos's watermarks possess important structural properties and are also meant to be embedded into the CFG of the software to be protected.[2] Though the mechanics of the proposed codec is well described in [1], the class of graphs that constitute the generated watermarks has not been fully characterized. Moreover, not much was known thus far about the resilience of Chroni and Nikolopoulos's graphs to malicious attacks, even though their ability to withstand attacks in the form of a single edge modification has been suggested without proofs. A thorough scrutiny of the structural properties of the aforementioned graph class allowed us to give it a formal characterization, as well as to introduce a linear-time decoding algorithm that retrieves the correct, untampered with encoded key even when $k \leq 2$ edges of the watermark are missing. Such algorithm allows for the determination of the exact resilience level of such graphs against distortive attacks.

The paper is organized as follows. In Section 2, we recall the codec from [1], formulating and proving a number of properties of the employed structures. In Section 3, we characterize the class of canonical reducible permutation graphs. In Section 4, we tackle the edge-removal attack model, proving that, for keys of size $n > 2$, it is always possible to identify and recover from attacks that remove $k \leq 2$ edges. Finally, in Section 5, we formulate a linear-time algorithm that reconstructs the original digraph, in case $k \leq 2$ edges are missing, recovering the encoded key thereafter. As a corollary of the results hitherto presented, we fully determine the resilience of the considered watermarking scheme. Section 6 concludes the paper with our final remarks and future directions. Throughout the paper, many proofs were omitted due to space constraints, and shall be included in a future extended version of the text.

2 The watermark by Chroni and Nikolopoulos

We start by briefly recalling the watermarking codec described in [1].

[1] A series of papers on watermarking by the same authors include, but is not limited to, [2–5].

[2] It is not in the scope of this paper the discussion of techniques to embed the watermark graph into a CFG.

Let ω be a positive integer key, with n the size of the binary representation B of ω. Let also n_0 and n_1 be the number of 0's and 1's, respectively, in B, and let f_0 be the index[3] of the leftmost 0 in B. The extended binary B^* is obtained by concatenating n digits 1, followed by the one's complement[4] of B and by a single digit 0. We let $n^* = 2n + 1$ denote the size of B^*, and we define $Z_0 = (z_i^0)$, $i = 1, \ldots, n_1 + 1$, as the ascending sequence of indexes of 0's in B^*, and $Z_1 = (z_i^1)$, $i = 1, \ldots, n + n_0$, as the ascending sequence of indexes of 1's in B^*.

Let S be a sequence. We denote by S^R the sequence formed by the elements of S in backward order. If $S = (s_i)$, $i = 1, \ldots, t$, is a sequence of size t, and there is an integer $k \leq t$ such that the subsequence consisting of the elements of S with indexes less than or equal to k is ascending, and the subsequence consisting of the elements of S with indexes greater than or equal to k is descending, then we say S is *bitonic*. If all t elements of a sequence S are distinct and belong to $\{1, \ldots, t\}$, then S is a *permutation*. If S is a permutation of size t, and, for all $1 \leq i \leq t$, the equality $i = s_{s_i}$ holds, then we say S is *self-inverting*. In this case, the unordered pair (i, s_i) is called a 2-*cycle* of S, if $i \neq s_i$, and a 1-*cycle* of S, if $i = s_i$. If S_1, S_2 are sequences, we denote by $S_1||S_2$ the sequence formed by the elements of S_1 followed by the elements of S_2.

Back to Chroni and Nikolopoulos's codec, the bitonic permutation $P_b = Z_0||Z_1^R = (b_i)$, $i = 1, \ldots, n^*$, is obtained by appending to Z_0 the elements of Z_1 in backward order, and, finally, the self-inverting permutation P_s is obtained from P_b as follows: for $i = 1, \ldots, n^*$, index b_i in P_s is assigned to element $s_{b_i} = b_{n^*-i+1}$, and index b_{n^*-i+1} in P_s is assigned to element $s_{b_{n^*-i+1}} = b_i$. In other words, the 2-cycles of P_s correspond to the n unordered pairs of distinct elements of P_b that are equidistant from the extremes of P_b, namely the pairs $(p, q) = (b_i, b_{n^*-i+1})$, for $i = 1, \ldots, n$. Since the central index $i = n + 1$ of P_b is the solution of equation $n^* - i + 1 = i$, element b_{n+1}—and no other—will constitute a 1-cycle in P_s. We refer to such element of P_s as its *fixed element*, and we let f denote it.

The watermark generated by the codec from [1] belongs to the class of reducible permutation graphs first defined in [7]. It is a directed graph G whose vertex set is $\{0, 1, \ldots, 2n + 2\}$, and whose edge set contains $4n + 3$ edges, to wit: a *path edge* $(u, u - 1)$ for $u = 1, \ldots, 2n + 2$, constituting a Hamiltonian path that will be unique in G, and a *tree edge* from u to $q(u)$, for $u = 1, \ldots, n^*$, where $q(u)$ is defined as the vertex $v > u$ with the greatest index in P_s to the left of u, if such v exists, or $2n + 2$ otherwise[5]. A graph so obtained is called a *canonical reducible permutation graph*.

Let us glance at an example. For $\omega = 43$, we have $B = 101011$, $n = 6$, $n_0 = 2$, $n_1 = 4$, $f_0 = 2$, $B^* = 1111110101000$, $n^* = 13$, $Z_0 = (7, 9, 11, 12, 13)$, $Z_1 = (1, 2, 3, 4, 5, 6, 8, 10)$, $P_b = (7, 9, 11, 12, 13, 10, 8, 6, 5, 4, 3, 2, 1)$, $P_s = (7, 9, 11, 12, 13, 10, 1, 8, 2, 6, 3, 4, 5)$ and $f = 8$. The generated watermark G will have, along with the path edges in Hamiltonian path $14 \to 13 \to \cdots \to 0$, also the tree edges $(1, 10)$, $(2, 8)$, $(3, 6)$, $(4, 6)$, $(5, 6)$, $(6, 8)$, $(7, 14)$, $(8, 10)$, $(9, 14)$, $(10, 13)$, $(11, 14)$, $(12, 14)$ and $(13, 14)$, as illustrated in Figure 1.

Canonical reducible permutation graphs are certainly not the only graphs that could be used to encode an integer, and they are certainly not the only graphs that could be disguised into a software's CFG. Yet they do have such properties, hence they are an appropriate choice. Moreover, they present some structural, encoding-related redundancy that grants them some resilience against attacks, as we shall see. We now state a number of properties of these graphs.

Let G still be the canonical reducible permutation graph associated to a key ω of size n,

[3]The index of the leftmost element in all sequences considered in the text is 1.

[4]The *one's complement* of a binary R is obtained by swapping all 0's in R for 1's and vice-versa.

[5]The rationale behind the name *tree edge* is the fact that such edges induce a spanning tree of $G \setminus \{0\}$.

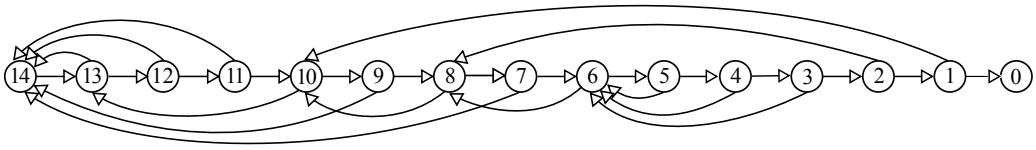

Figure 1: Watermark for key $\omega = 43$.

and P_b and P_s, respectively, the bitonic and the self-inverting permutations dealt with during the construction of G.

Property 1 *For $1 \leq i \leq n$, element b_{n+i+1} in P_b is equal to $n - i + 1$, that is, the n rightmost elements in P_b are $1, 2, \ldots, n$ when read from right to left.*

Property 2 *The elements whose indexes are $1, 2, \ldots, n$ in P_s are all greater than n.*

Property 3 *The fixed element f satisfies $f = n + f_0$, unless the key ω is equal to $2^k - 1$ for some integer k, whereupon $f = n^* = 2n + 1$.*

Property 4 *In self-inverting permutation P_s, elements indexed $1, 2, \ldots, f - n - 1$ are respectively equal to $n+1, n+2, \ldots, f-1$, and elements indexed $n+1, n+2, \ldots, f-1$ are respectively equal to $1, 2, \ldots, f - n - 1$.*

Property 5 *The first element in P_s is $s_1 = n + 1$, and the central element in P_s is $s_{n+1} = 1$.*

Property 6 *If $f \neq n^*$, then the index of element n^* in P_s is equal to $n_1 + 1$, and vice-versa. If $f = n^*$, then the index of element n^* in P_s is also n^*.*

Property 7 *The subsequence of P_s consisting of elements indexed $1, 2, \ldots, n + 1$ is bitonic.*

Property 8 *For $u \neq 2n + 1$, $(u, 2n + 2)$ is a tree edge of watermark G if, and only if, $u - n$ is the index of a digit 1 in the binary representation B of the key ω represented by G.*

Property 9 *If (u, k) is a tree edge of watermark G, with $k \neq 2n + 2$, then (i) element k precedes u in P_s; and (ii) if v is located somewhere between k and u in P_s, then $v < u$.*

3 Canonical reducible permutation graphs

This section is devoted to the characterization of the class of canonical reducible permutation graphs. After describing some terminology, we define the class using purely graph-theoretical predicates, then we prove it corresponds exactly to the set of all watermark instances possibly produced by Chroni and Nikolopoulos's encoding algorithm [1]. Finally, we characterize it in a way that suits the design of a new, resilient, linear-time decoding algorithm.

Given a graph G, we let $V(G)$ and $E(G)$ denote, as usual, the vertex set and edge set of G, respectively. Also, we let $N_G^+(v)$ and $N_G^-(v)$ respectively denote the set of out-neighbors and in-neighbors of vertex $v \in V(G)$. A *reducible flow graph* [11, 12] is a directed graph G with source $s \in V(G)$, such that, for each cycle C of G, every path from s to C reaches C at a same vertex. It is well known that a reducible flow graph has at most one Hamiltonian cycle.

Definition 10 *A* self-labeling reducible flow graph *relative to $n > 1$ is a directed graph G s.t.*
(i) $|V(G)| = 2n + 3$;

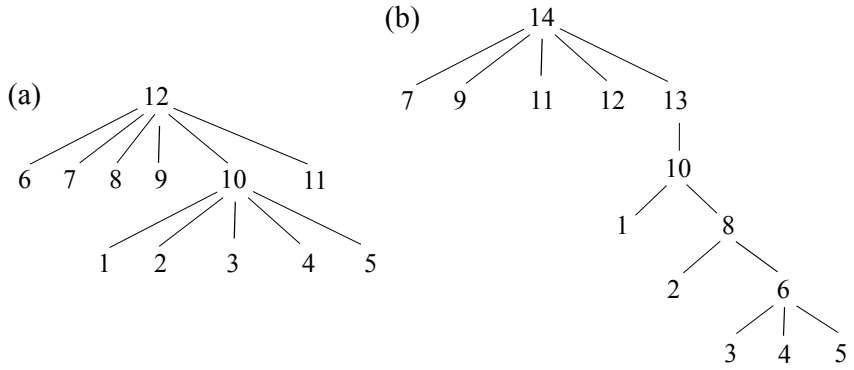

Figure 2: Representative trees of the watermarks for keys (a) $\omega = 31$ and (b) $\omega = 43$.

(ii) *G presents exactly one directed Hamiltonian path, hence there is a unique labeling function $\sigma : V(G) \to \{0, 1, \ldots, 2n+2\}$ of the vertices of G such that the order of the labels along the Hamiltonian path is precisely $2n+2, 2n+1, \ldots, 0$;*

(iii) *considering the labeling σ as in the previous item, $N_G^+(0) = \emptyset$, $N_G^-(0) = \{1\}$, $N_G^+(2n+2) = \{2n+1\}$, $|N_G^-(2n+2)| \geq 2$, and, for all $v \in V(G) \setminus \{0, 2n+2\}$, $N_G^+(v) = \{v-1, w\}$, for some $w > v$.*

From now on, without loss of generality, we shall take σ for granted and assume the vertex set of any self-labeling reducible flow graph G *is* the very set $V(G) = \{0, 1, \ldots, 2n+2\}$, yielding the unique Hamiltonian path $2n+2, 2n+1, \ldots, 0$ in G.[6]

Let G be a self-labeling reducible flow graph and H its unique Hamiltonian path. We define a tree T with vertex set $V(T) = V(G) \setminus \{0\}$, and edge set $E(T)$ comprising all edges in $E(G) \setminus E(H)$ deprived of their orientation. We call T the *representative tree* of G, and we regard it as a *rooted* tree whose root is $2n+2$, and where the children of each $v \in V(T)$, denoted $N_T(v)$, are exactly the in-neighbors of v in $G \setminus E(H)$. In addition, we regard T as an *ordered* tree, that is, for each $v \in V(T)$, the children of v are always considered according to an ascending order of their labels. Finally, for $v \in T$, we denote by $N_T^*(v)$ the set of descendants of v in T. Figure 2 depicts two representative trees.

Observation 11 *The representative tree T of a self-labeling reducible flow graph G satisfies the max-heap property, that is, if vertex u is a child of vertex v in T, then $v > u$.*

We convey the idea that a representative tree T satisfies the max-heap property by saying that T is a *descending*, ordered, rooted tree.

Definition 12 *A self-inverting permutation S of size $2n+1$ is said to be* canonical *if:*

(i) *there is exactly one 1-cycle in S;*

(ii) *each 2-cycle (s_i, s_j) of S satisfies $1 \leq i \leq n$, for $s_i > s_j$;*

(iii) *$s_1, \ldots, s_{n+1}$ is a bitonic subsequence of S starting at $s_1 = n+1$ and ending at $s_{n+1} = 1$.*

[6]By doing so, we may simply compare two vertices, e.g. $v > u$ (or v greater than u, in full writing), whereas we would otherwise need to compare their images under σ, e.g. $\sigma(v) > \sigma(u)$.

Lemma 13 *In any canonical self-inverting permutation of* $\{1, \ldots, 2n+1\}$*, the fixed element* f *satisfies* $f \in [n+2, 2n+1]$.

Let T be some representative tree, therefore a descending, ordered, rooted tree. The *preorder traversal* P of T is a sequence of its vertices that is recursively defined as follows. If T is empty, P is also empty. Otherwise, P starts at the root r of T, followed by the preorder traversal of the subtree whose root is the first (i.e. smallest) child of r, followed by the preorder traversal of the subtree whose root is the second smallest child of r, and so on. The last (rightmost) element of P is referred to as the rightmost element of T as well.

Lemma 14 *The preorder traversal of a representative tree* T *is unique. Conversely, a representative tree* T *is uniquely determined by its preorder traversal.*

If we remove the first element of P, the remaining sequence is said to be the *root-free* preorder traversal of T.

Definition 15 *A canonical reducible permutation graph* G *is a self-labeling reducible graph such that the root-free preorder traversal of the representative tree of* G *is a canonical self-inverting permutation.*

Theorem 16 *A digraph* G *is a watermark instance produced by Chroni and Nikolopoulos's encoding algorithm [1] if, and only if,* G *is a canonical reducible permutation graph.*

Let T be the representative tree of some canonical reducible permutation graph G, and P a canonical self-inverting permutation corresponding to the root-free preorder traversal of T. We refer to the fixed element f of P also as the fixed element (or vertex) of both G and T. Similarly, the 2-cyclic elements of P correspond to *cyclic* elements (or vertices) of both G and T. The concepts we describe next will be employed in the characterization of canonical reducible permutation graphs.

A vertex $v \in V(T) \setminus \{2n+2\}$ is considered *large* when $n < v \leq 2n+1$; otherwise, $v \leq n$ and v is dubbed as *small*. Denote by X, Y, respectively, the subsets of large and small vertices in T, so $|X| = n+1$ and $|Y| = n$. By Lemma 13, $f \in X$. We then define $X_c = X \setminus \{f\} = \{x_1, \ldots, x_n\}$ as the set of large cyclic vertices in T.

We say that T is a *type*-1 tree—please see Figure 3(a)—when

(i) $n+1, n+2, \ldots, 2n+1$ are children of the root $2n+2$ in T; and

(ii) $1, 2, \ldots, n$ are children of $2n$.

Elseways, we say that T is a *type*-2 tree relative to f— please see Figure 3(b)—when

(i) $n+1 = x_1 < x_2 < \ldots < x_\ell = 2n+1$ are the children of $2n+2$, for some $\ell \in [2, n-1]$;

(ii) $x_i > x_{i+1}$ and x_i is the parent of x_{i+1}, for all $i \in [\ell, n-1]$;

(iii) $1, 2, \ldots, f-n-1$ are children of x_n;

(iv) $x_i = n+i$, for $1 \leq i \leq f-n-1$;

(v) f is a child of x_q, for some $q \in [\ell, n]$ satisfying $x_{q+1} < f$ whenever $q < n$; and

(vi) $N_T^*(f) = \{f-n, f-n+1, \ldots, n\}$ and $y_i \in N_T^*(f)$ has index $x_{y_i} - f + 1$ in the preorder traversal of $N_T^*[f]$.

Lemma 17 *If y_r is the rightmost vertex of a type-2 representative tree T relative to some $f \neq 2n+1$, then $y_r = \ell$.*

The following theorem characterizes canonical reducible permutation graphs.

Theorem 18 *A digraph G is a canonical reducible permutation graph if, and only if, G is a self-labeling reducible graph and*

(i) the fixed element of G is $2n+1$ and G has a type-1 representative tree; or

(ii) the fixed element of G belongs to $[n+2, 2n]$ and G has a type-2 representative tree.

4 Restoring a damaged watermark

In this section, we analyze the effects of a distortive attack against a canonical reducible permutation graph G whereby two edges $e_1, e_2 \in E(G)$ were removed. Let $G' = G \setminus \{e_1, e_2\}$.

4.1 Reconstructing the Hamiltonian path

The algorithm given in pseudocode as Procedure 1 reconstructs the Hamiltonian path H of G, in case e_1 or e_2 belonged to H, and classifies each missing edge as either a path edge or a tree edge. The input is the damaged watermark graph G'. If $v \in V(G')$, we denote by $H(v)$ the subsequence of the Hamiltonian path of G that ends at v and starts as far as possible in G'. Also, we denote by $first(H(v))$ the first vertex of the subsequence $H(v)$.

The mechanics of Procedure 1 is that of sewing together the $k' \leq 3$ maximal directed paths resulting from the deletion of $k \leq 2$ edges from G. In short, each such directed path is reassembled by placing vertices, one by one in backwards fashion, starting at a vertex with out-degree 0 among those which have not yet been placed. The proof that it is always possible to restore the Hamiltonian path this way relies on the characterization of representative trees and is not overly complicated. It is, however, quite lengthy, since each possible case (see Figure 4) resulting from such an attack must be tackled separately. It has therefore been omitted.

As for the time complexity of the algorithm, note that, in general, $1 \leq |V_0| \leq 2$ and $1 \leq |V_1| \leq 3$. The latter follows from the definition of a self-labeling reducible graph and from the fact that two edges were removed from G. Moreover, each path $H(v_i)$ can be computed in $\mathcal{O}(|H(v_i)|)$ time. Consequently, the entire algorithm has complexity $\mathcal{O}(n)$.

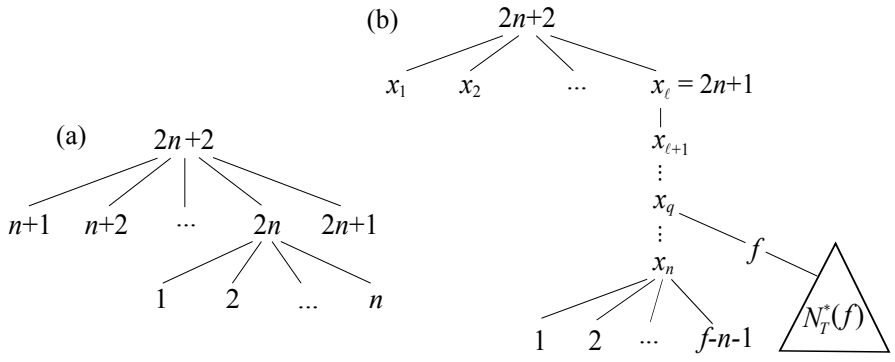

Figure 3: (a) A type-1 representative tree. (b) A type-2 representative tree.

Procedure 1: Reconstructing the Hamiltonian path

$V_0 \leftarrow \{v \in V(G') \text{ s.t. } |N^+_{G'}| = 0\}$; $V_1 \leftarrow \{v \in V(G') \text{ s.t. } |N^+_{G'}| = 1\}$
if $|V_0| = 1$ **then**
  let v_0 be the unique element in V_0
  if $|H(v_0)| = 2n+3$ **then** $H \leftarrow H(v_0)$, **return** H and edge types as in Figure 4(a)
  else if $\exists\ v_1 \in V_1$ such that $|H(v_0)| + |H(v_1)| = 2n+3$ **then**
    $H \leftarrow H(v_1)||H(v_0)$, **return** H and edge types as in Figures 4(b,c)
  else
    let $v_1, v_1' \in V_1$ be such that
      $|H(v_0)| + |H(v_1)| + |H(v_1')| = 2n+3$ and $N^+_{G'}(first(H(v_1)) \cap H(v_1') \neq \emptyset$
    $H \leftarrow H(v_1')||H(v_1)||H(v_0)$, **return** H and edge types as in Figure 4(d)
else
  let v_0, v_0' be the elements in V_0
  if $|H(v_0)| + |H(v_0')| = 2n+3$ **then**
    let v_0 be such that $N^+_{G'}(first(H(v_0))) \cap H(v_0') \neq \emptyset$
    $H \leftarrow H(v_0')||H(v_0)$, **return** H and edge types as in Figures 4(e,f)
  else
    let $v_0' \in V_0$ and $v_1 \in V_1$ be such that $v_0' \in N^+_{G'}(first(H(v_1))$
    $H \leftarrow H(v_0')||H(v_1)||H(v_0)$, **return** H and edge types as in Figures 4(g,h)

4.2 Determining the fixed element

Suppose the watermark G has been attacked, which resulted in a damaged watermark G', where two unknown edges are missing. Now we shall recognize the fixed element of the original watermark, given the damaged one. Getting to know the fixed element of G will play a crucial role in retrieving the missing tree edges and consequently restoring the original key w.

The two following theorems, whose proofs we omit, characterize the fixed element f of G when $f = 2n+1$ and when $f < 2n+1$, respectively. Let T be the representative tree of the original watermark G. We consider the case where the two edges that have been removed belong to T. Denote by F the forest obtained from T by the removal of two edges.

Theorem 19 *Let F be a forest obtained from the representative tree T by removing two edges, where $n > 2$. The fixed element of T is $f = 2n+1$ if, and only if,*

(i) vertex $2n+1$ is a leaf of F; and

(ii) the n small vertices of G' are children of $2n$ in F, with the possible exception of at most two of them, in which case they must be isolated vertices.

Theorem 20 *Let F be a forest obtained from the representative tree T of watermark G by removing two of its edges, and let $x \leq 2n$ be a large vertex of T which is not a child of $2n+2$. Vertex x is the fixed element f of G if, and only if,*

(i) the large vertex x has a sibling z in F, and $x > z$; or

(ii) the subset of small vertices $Y' \subset Y$, $Y' = \{x-n, x-n+1, \ldots, n\}$ can be partitioned into at most two subsets Y_1', Y_2', such that $\emptyset \neq Y_1' = N^+_F(x)$ and Y_2' is the vertex set of one of the trees which form F; or, whenever the previous conditions do not hold,

(iii) the large vertex x is the rightmost vertex of one of the trees of F, while the rightmost vertices of the remaining trees are all small vertices.

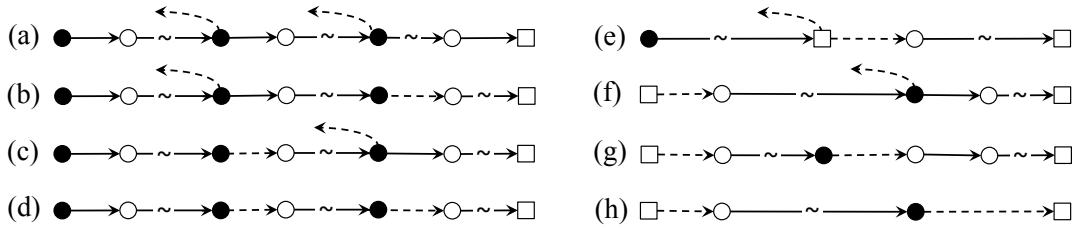

Figure 4: Possible scenarios for the Hamiltonian path of a damaged watermark G. Dashed arrows indicate missing edges. Broken arrows (with a tilde in the middle) indicate paths of arbitrary size $d \geq 0$ (i.e., both ends may coincide). Squares, solid circles and hollow circles represent vertices whose out-degrees in G' are, respectively, 0, 1 and 2. The tree edges of G' are not being shown (unless those removed by the attacker, represented by backward arrows).

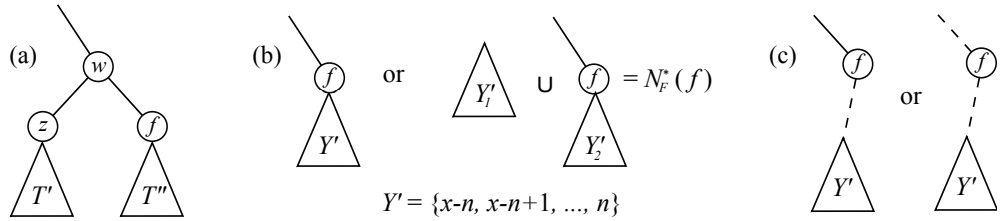

Figure 5: (a–c) Conditions (i), (ii) and (iii) of Theorem 20, respectively.

The above theorems lead to an algorithm that finds the fixed element of G in linear time. The input is the forest F, obtained from the representative tree T of G by the removal of two edges. The algorithm is as follows: first, check whether $f = 2n + 1$. This is a direct task, applying Theorem 19: just verify if $2n + 1$ is a leaf of F and whether all small vertices are children of $2n$, except possibly two, which must be isolated vertices. If both conditions are satisfied then set $f = 2n + 1$ and terminate the algorithm. Otherwise, proceed to determining $f < 2n + 1$ by checking the conditions described in Theorem 20 (see also Figure 5).

4.3 Determining the root's children

After having identified the fixed element of the watermark, we are almost in a position to determine the tree edges that have been removed.

Observe that, when $f = 2n+1$, the task is trivial, since, in this case, by Theorem 18, there can be only one canonical reducible permutation graph G relative to n. Such a graph is precisely the one with a type-1 representative tree T, which is unique for each $n > 2$ (cf. Property 3 of canonical reducible permutation graphs, in Section 2). By definition, the root-free preorder traversal of a type-1 representative tree, when $f = 2n+1$, is $n+1, n+2, \ldots, 2n, 1, 2, \ldots, n, 2n+1$.

We therefore want to determine the children of $2n+2$ restricted to the case where $f < 2n+1$. Let G be a watermark, T its representative tree and F the forest obtained from T by the removal of two edges. As usual, f stands for the fixed element of T, X is the set of large vertices other than $2n + 2$, and $X_c = X \setminus \{f\}$. Finally, denote by $A \subseteq X_c$ the subset of ascending large cyclic vertices of T, which we shall refer to simply as the *ascending* vertices, and denote by D the set $D = X_c \setminus A$ of descending large cyclic vertices of T, or simply the *descending* vertices.

Procedure 2: Finding $f \neq 2n+1$

1. If F contains a large vertex x having a sibling z
   then let $f \leftarrow max\{x, z\}$ and terminate the algorithm. Otherwise,
2. For each large vertex x of F satisfying $N_F(x) \neq \emptyset$ and each small $y \in N_F(x)$,
   let $Y' = \{x-n, x-n+1, \ldots, n\}$. If $N_F^*(x) = Y'$ or $N_F^*(x) \subset Y'$,
   and $Y' \setminus N_F^+(x)$ is the vertex set of one of the trees of F,
   then let $f \leftarrow x$ and terminate the algorithm. Otherwise,
3. Find the preorder traversals of the three trees of F, and
   then let f be the unique vertex that is both large and the rightmost element
   of the preorder traversal of some tree of F.

Procedure 3: Constructing the set of large ascending vertices

1. If $F[X_c] \cup \{2n+2\}$ is connected then $A \leftarrow N_F(2n+2)$
   and terminate the algorithm. Otherwise,
2. If $F[X_c] \cup \{2n+2\}$ contains no isolated vertices then $A \leftarrow N_F(2n+2) \cup \{2n+1\}$
   and terminate the algorithm. Otherwise,
3. If $F[X_c] \cup \{2n+2\}$ contains two isolated vertices x, x' then $A \leftarrow N_F(2n+2) \cup \{x, x'\}$
   and terminate the algorithm. Otherwise,
4. If $F[X_c] \cup \{2n+2\}$ contains a unique isolated vertex x then
   if $|N_F^*(f)| = 2n - f + 1$ then
   let y_r be the rightmost vertex of $N_F^*(f)$
   if $|N_F(2n+2)| < y_r$ then $A \leftarrow N_F(2n+2) \cup \{x, 2n+1\}$
   else $A \leftarrow N_F(2n+2)$
   else $A \leftarrow N_F(2n+2) \cup \{x\}$

Given the forest F and its fixed element f, Procedure 3 computes the set A, which, due to the representative tree properties of T, corresponds precisely to the children of its root $2n+2$. The notation still employs $N_F(v)$ and $N_F^*(v)$ for the children and the descendants of vertex v in F, respectively. We denote by $F[X_c]$ the subgraph of F induced by the vertices in X_c.

Theorem 21 *Procedure 3 correctly computes the set of ascending vertices of T in linear time.*

Once we manage to recover the set A of children of the root $2n+2$ in T, the decoding algorithm proposed in the next section can be run, retrieving the encoded key ω. This suffices to prove that the original, undamaged watermark can be fully restored: a simple possibility is to run the (linear) encoding algorithm from scratch, with ω as input. We have, however, an even simpler algorithm that restores the watermark based on the reconstitution of its preorder traversal, but we shall not present it in the present paper due to space constraints.

5 A new decoding algorithm

We can now formulate our new decoding algorithm. If the input watermark presents $k \leq 2$ missing edges, the algorithm is able to fix it, prior to running the decoding step. The decoding

step itself is absolutely straightforward, and relies on the following theorem.[7]

Theorem 22 *Let ω be a given key and G the watermark corresponding to ω. Let $A' = x_1, \ldots, x_{\ell-1}$ be the ascending sequence of children of $2n+2$, in the representative tree T of G, that are different from $2n+1$. Then $\omega = \sum_{x_i \in A'} 2^{2n-x_i}$.*

As a consequence of the above theorem, whenever the input watermark has *not* been tampered with, the proposed algorithm simply retrieves the encoded key in a faster, less complicated fashion than the original decoding algorithm from [1].[8]

Algorithm 4: Obtaining the key from a possibly damaged watermark

1. Let $k \leftarrow |E(G)| - (4n+3)$.
2. If $k > 2$, report the occurrence of k edge removals and halt.
3. If $0 < k \leq 2$, proceed to the reconstitution (Procedures 1–3).
4. Calculate and return the key ω as indicated by Theorem 22.

Theorem 23 *Algorithm 4 retrieves the correct key $\omega \geq 4$, encoded in a watermark with up to two missing edges, in linear time.*

Corollary 24 *Distortive attacks in the form of k edge modifications (insertions/deletions) against canonical reducible permutation graphs G, with $|V(G)| = 2n+3$, $n > 2$, can be detected in polynomial time, if $k \leq 5$, and also recovered from, if $k \leq 2$. Such bounds are tight.*

6 Final considerations

After characterizing the class of canonical reducible permutation graphs, we formulated a linear-time algorithm that restores a member of that class presenting up to two missing edges. Our results therefore have proved that canonical reducible permutation graphs are always able to detect and recover from malicious attacks in the form of $k \leq 2$ edge removals[9] in linear time. Moreover, we have shown that attacks in the form of $k \leq 5$ edge modifications (insertions/deletions) can be detected in polynomial time. Such level of resilience, we remark, is a very important feature of the original watermarking scheme.

Future directions. A necessary condition for a watermark G_1 to recover from the removal of a subset of edges $E_1' \subset E(G_1)$, with $|E_1'| = k$, is that $G_1' = G_1 \setminus E_1'$ is not isomorphic to some graph G_2' obtained from watermark $G_2 \neq G_1$ by the removal of k edges. For $k \leq 2$, we have shown this condition is always satisfied, provided $n > 2$, and we have proved this is not always true for $k \geq 3$. An interesting open problem is therefore to characterize the set of keys $\Omega(k)$ whose corresponding watermarks can always recover from the removal of $k \geq 3$ edges.

Future research focusing on the development of watermarking schemes resilient to attacks of greater magnitude may consider extending the concept of canonical reducible permutation graphs by allowing permutations with h-cycles, with $h > 2$, as well as multiple fixed elements.

[7] Though some reconstitution steps were written as if *exactly* two edges were missing, the case where a single edge is missing is at least as easy, since removing an arbitrary edge transforms the latter case into the former.

[8] Note that, in this case, it is straightforward to determine the set A, as $A = N_G^-(2n+2)$.

[9] The sole exceptions are two very small instances G, G' corresponding to keys of size $n = 2$, namely keys $\omega = 2$ (binary $B = 10$) and $\omega' = 3$ (binary $B = 11$), respectively, which become isomorphic when edges $(1,5),(4,5)$ are removed from G_1 and edges $(1,4),(4,6)$ are removed from G_2.

References

[1] M. Chroni and S.D. Nikolopoulos, Efficient encoding of watermark numbers as reducible permutation graphs, arXiv:1110.1194v1 [cs.DS], 2011.

[2] M. Chroni and S.D. Nikolopoulos, Encoding watermark numbers as cographs using self-inverting permutations, *12th Int'l Conference on Computer Systems and Technologies, CompSysTech'11*, ACM ICPS **578** (2011), 142–148 (Best Paper Award).

[3] M. Chroni and S.D. Nikolopoulos, An efficient graph codec system for software watermarking, *36th IEEE Conference on Computers, Software, and Applications (COMPSAC'12)*, IEEE Proceedings (2012), 595–600.

[4] M. Chroni and S.D. Nikolopoulos, Multiple encoding of a watermark number into reducible permutation graphs using cotrees, *13th Int'l Conference on Computer Systems and Technologies (CompSysTech'12)*, ACM ICPS Proceedings (2012), 118–125.

[5] M. Chroni and S.D. Nikolopoulos, An embedding graph-based model for software watermarking, *8th Int'l Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIH-MSP' 12*, IEEE Proceedings (2012), 261–264.

[6] C. Collberg, A. Huntwork, E. Carter, G. Townsend and M. Stepp, More on graph theoretic software watermarks: implementation, analysis and attacks, *Information and Software Technology* **51** (2009), 56–67.

[7] C. Collberg, S. Kobourov, E. Carter and C. Thomborson, Error-correcting graphs for software watermarking, *29th Workshop on Graph-Theoretic Concepts in Computer Science, WG'03*, LNCS **2880** (2003), 156–167.

[8] C. Collberg and C. Thomborson, Software watermarking models and dynamic embeddings, *Proc. 26th ACM SIGPLAN-SIGACT on Principles of Programming Languages, POPL'99* (1999), 311–324.

[9] C. Collberg, C. Thomborson and G. M. Townsend, Dynamic graph-based software fingerprinting, *ACM Transactions on Programming Languages and Systems* **29** (2007), 1–67.

[10] R. L. Davidson and N. Myhrvold, Method and system for generating and auditing a signature for a computer program, US Patent 5.559.884, Microsoft Corporation (1996).

[11] M. S. Hecht and J. D. Ullman, Flow graph reducibility, *SIAM J. Computing* **1** (1972), 188–202.

[12] M. S. Hecht and J. D. Ullman, Characterizations of reducible flow graphs, *Journal of the ACM* **21** (1974), 367–375.

[13] J. Zhu, Y. Liu and K. Yin, A novel dynamic graph software watermark scheme, *1st Int'l Workshop on Education Technology and Computer Science* **3** (2009), 775–780.

[14] R. Venkatesan, V. Vazirani, S. Sinha, A graph theoretic approach to software watermarking, *4th International Information Hiding Workshop* (2001), 157–168.