Software, Image and Audio Watermarking Algorithmic Techniques

Ioannis Chionis Maria Chroni Angelos Fylakis Stavros D. Nikolopoulos
Department of Computer Science and Engineering, University of Ioannina
GR-45110 Ioannina, Greece
{ichionis, mchroni, afylakis, stavros}@cs.uoi.gr

***Abstract*—Digital watermarking involves embedding a watermark value within a digital object, such as image, audio, video, text and software, to prove authenticity in case of intellectual property infringement. Headed to this direction, in this paper we survey our previous algorithmic techniques for software and image watermarking and present a new developing idea based on them for audio watermarking. Our watermarking techniques take as an input a watermark that is an integer w which can be efficiently encoded as a self-inverting permutation π^*. We demonstrate multiple representations of self-inverting permutations, namely reducible permutation graphs, two-dimensional and one-dimensional matrices. We propose efficient algorithmic techniques for watermarking software, image and audio that exploit self-inverting permutation representations in order to embed the watermark w by making imperceptible modifications and producing equivalent watermarked objects of high fidelity.**

I. Introduction

The last few years digital communication has become an indispensable part for everyday life since most people use it on a regular basis and do many daily activities online. This frequent use of the world wide web raises issues concerning security measures during internet usage since the web is not risk-free. One of those risks is the fact that the web is an environment where intellectual property is under threat since a huge amount of public personal data is continuously transferred, and thus such data may end up on a user who falsely claims ownership. And that is where watermarks come into place.

The idea of digital watermarking arose independently in the year 1990 on an image watermarking study and it was around 1993 when Tirkel et al. [14] introduced the word "water mark" which became "watermark" later on, while the first software watermarking technique was presented in 1996 by Davidson and Myhrvold [10]. Watermarks are symbols which are placed into physical objects and their purpose is to carry information about an object's authenticity. In our case the watermarks have digital form and they are embedded into digital objects; this technique is called digital watermarking. Digital watermarking is a technique for protecting the intellectual property of a digital object; the idea is simple: a unique identifier, which is called *watermark*, is embedded into a digital object which may be used to verify its authenticity or the identity of its owners [9]. A digital object may be audio, picture, video, or software, and the watermark is embedded into the object's data in such a way that it is not detectable by human perception.

ISCIM 2013, pp. 29-34

In this paper we present algorithmic techniques for watermarking software, image and audio through the exploitation of graph theoretical objects. Specifically those objects are the three alternative representations of self-inverting permutations that we suggest for hiding information in each case.

It is fair to point out that the main idea of this paper along with a variety of algorithmic techniques for software and image watermarking have been presented in several fora by the authors during the last three years [2], [3], [4], [5], [6], [7]. Based on the same idea we here extent the pool of the physical objects that can be efficiently watermarked by proposing an algorithmic technique for audio watermarking.

II. Watermark Encodings

In this section we describe the discrete structures that are used to encode a watermark number, namely permutations and self-inverting permutations. We then briefly present the different transformations of the self-inverting permutation into a reducible permutation graph, a two-dimensional matrix and one-dimensional matrix which are the key components that are used to embed the watermark into any digital object i.e. software, image and audio in our watermarking models.

A. Encoding Numbers as SiP

A watermark integer w can be converted to a self-inverting permutation in several ways [13]. We have proposed an algorithm which efficiently encodes an integer w into a self-inverting permutation π and efficiently decodes it through the use of bitonic permutations [2].

Definition 1. Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ be a permutation over the set $N_n = \{1, 2, \ldots, n\}$, where $n > 1$. The inverse of the permutation π is the permutation $q = (q_1, q_2, \ldots, q_n)$ with $q_{\pi_i} = \pi_{q_i} = i$. A *self-inverting permutation* (or, for short, SiP) is a permutation that is its own inverse: $\pi_{\pi_i} = i$.

By definition, a permutation is a self-inverting permutation (SiP) if and only if all its cycles are of length 1 or 2. Figure 1 shows the SiP $\pi = (4, 7, 6, 1, 5, 3, 2)$ of the watermarked number $w = 4$ which consists of cycles (1,4), (2,7), (3,6) and (5). Hereafter, we shall denote a SiP by π^*.

We show below by an example the encoding of the watermark integer 4 into the corresponding SiP π^*; our algorithm's encoding process is described in [2] (see also [1]).

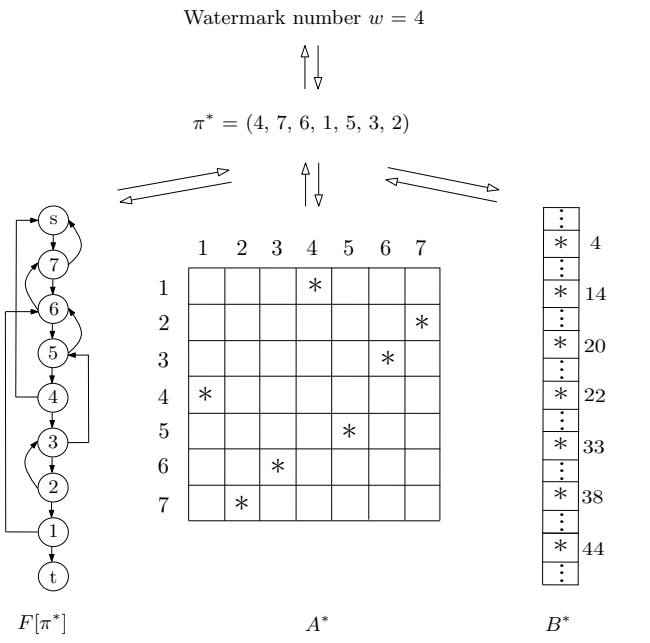

Fig. 1. The main data components used by our codec algorithms for software, image and audio watermarking.

An example of encoding w to π^*. Let $w = 4$ be the input watermark integer. We first compute the binary representation $B = 100$ of the number 4; then we construct the binary number $B' = 0001001$ by concatenating the binary numbers 000, 100 and 1, and the binary sequence $B^* = (1, 1, 1, 0, 1, 1, 0)$ by flipping the binary number B'; we compute the sequences $X = (4, 7)$ and $Y = (1, 2, 3, 5, 6)$ by locating the indices of 0's and 1's in B^*, and then construct the bitonic permutation $\pi = (4, 7, 6, 5, 3, 2, 1)$ on $n' = 7$ numbers; since n' is an odd number, we select 3 cycles $(4, 1)$, $(7, 2)$, $(6, 3)$ of length 2 and one cycle $(5, 5)$ of length 1, and then construct the self-inverting permutation $\pi^* = (4, 7, 6, 1, 5, 3, 2)$.

B. SiP Encodings

Having proposed in [2] the encoding of a watermark integer w into self-inverting permutation, we next present the different representations of a self-inverting permutation into a reducible permutation graph, a two-dimensional matrix and one-dimensional matrix.

(a) Reducible Permutation Graph (RPG)
We have proposed an efficient algorithm for encoding a self-inverting permutation into a reducible permutation flow-graph $F[\pi^*]$ by using an efficient directed acyclic graph (DAG) representation of the self-inverting permutation π^*. We also proposed the decoding algorithm which extracts the self-inverting permutation π^* from $F[\pi^*]$ by converting first the the graph $F[\pi^*]$ into a directed tree $T[\pi^*]$ and then applying DFS search on $T[\pi^*]$ [3] (see also [5]). The whole encoding process takes $O(n)$ time and requires $O(n)$ space, where n is the length of the input permutation π^*. Given a self-inverting permutation π^* of length n our encoding algorithm works on two phases:

(I) it first uses a strategy to transform the permutation π^* into a directed acyclic graph $D[\pi^*]$ using certain combinatorial properties of the elements of π^*;
(II) then, it constructs a directed graph $F[\pi^*]$ on $n+2$ nodes using the adjacency relation of the nodes of the DAG $D[\pi^*]$.

The reducible permutation graph $F[\pi^*]$ we construct is a directed graph on $n + 2$ nodes with outdegree exactly two, in order to match real program graphs, and with descending ordering on its nodes $V(G) = \{s = u_{n+1}, u_n, \ldots, u_1, u_0 = t\}$. Also, it has been proved that $F[\pi^*]$ has always a unique Hamiltonian path, and this path can be found on $O(n)$ time.

Figure 1 shows the reducible permutation graph $F[\pi^*]$ that corresponds to the watermark number $w = 4$.

(b) Two-dimensional Representation of SiP
We have designed an efficient algorithm to encode a self-inverting permutation $\pi^* = (\pi_1^*, \pi_2^*, \ldots, \pi_n^*)$ of size n into the two-dimensional space (2D representation) by mapping specific cells of an $n \times n$ matrix A. In fact what we do is to label the cell at row i and column π_i^* with the number π_i^*, for each $i = 1, 2, \ldots, n$. The 2D representation A of π^* is a square and symmetric array on its main diagonal. Since a self-inverting permutation has cycles of length two and only one cycle of length one (see Subsection II-A), the matrix A will always have only one element on its main diagonal. The symmetric property is important and allows the reconstruction of the self-inverting permutation in the case where an element has been deleted from the matrix.

We have also described the two-dimensional marked representation (2DM representation) of a permutation π^* based on the previously defined 2D representation of a permutation π^*. The 2DM representation of π^* can be applied in image watermarking and incorporates the properties of the 2D representation.

In our 2DM representation, a self-inverting permutation π^* is represented by an $n \times n$ matrix A^* as follows: if $A(i, j) = \pi^*$ then $A^*(i, j)$ cell is marked by a specific symbol, i.e., the character "*", $1 \leq i, j \leq n$.

Figure 1 illustrates the 2DM representation of the watermark number $w = 4$.

(c) One-dimensional Representation of SiP
A self-inverting permutation $\pi^* = (\pi_1^*, \pi_2^*, \ldots, \pi_n^*)$ of size n can be represented in the one-dimensional space (1D representation) by mapping specific cells of a matrix B of size n^2. In fact what we do is for each element π_i^* of the self-inverting permutation π^* we label the cell at position $(1, (i-1)n + \pi_i^*)$ of matrix B by the number π_i^*, where i is the position of element π_i^* in permutation π^*. Based on the 1D representation of the SiP we construct the one-dimensional marked representation (1DM representation) of π^* which we use in order to watermark audio signals.

In the proposed 1DM representation, the self-inverting permutation is represented by a one-dimensional matrix B^* of size n^2, where the cells of B^* are marked by a specific symbol,

such as the character "*" in our case. For each element i, $1 \leq i \leq n$, of SiP π^* the matrix B^* is formulated as follows: $B^*[(i-1)n + \pi_i^*]$="*".

Figure 1 illustrates the 1DM representation of the watermark number $w = 4$.

Remark 1. It is easy to see that the 1DM representation of π^*, can be constructed from its 2DM representation as follows: if cell(i, j) of A^* is marked by "*" then $B^*[(i-1)n+j]$="*", $0 \leq i, j \leq n$.

III. Software Watermarking

Having encoded a watermark number w as reducible permutation graph $F[\pi^*]$ (see Subsection II-B), we next propose a dynamic watermarking model for embedding the reducible permutation graph $F[\pi^*]$ into an application program P. Our model uses the dynamic call-graph $G(P, I_{key})$ of the program P, taken by the specific input I_{key}, and the reducible permutation graph $F[\pi^*]$, and produces the watermarked program P^* having the following key property: its dynamic call-graph $G(P^*, I_{key})$ and the reducible permutation graph $F[\pi^*]$ are isomorphic graphs. Within this idea the program P^* is produced by only altering appropriate real-calls of specific functions of the input program P.

The main idea of our embedding method is to alter the execution-flow of appropriate function calls of P such that the execution of the resulting program P^* with the input I_{key} produces a call-graph $G(P^*, I_{key})$ which after removing the node f_{main} is isomorphic to watermark graph $F[\pi^*]$.

In order to modify the execution flow of an application program P and retain the functionality of P we modify the functions that are executed with I_{key} by applying specific model components. The model components include the dynamic call-graph $G(P^*, I_{key})$, the call patterns, the control statements, and the execution rules, see [6] for more details. In fact for each function call (f_i, f_j) that is executed with I_{key} we either apply the components in f_i if the function call (f_i, f_j) corresponds to an edge in RPG, or we replace the call (f_i, f_j) by a sequence of calls with corresponding edges in the RPG, forming a shortest path, and then we apply the components.

Figure 2 shows the dynamic call-graph $G(P, I_{key})$ of an application program P, the reducible permutation graph $F[\pi^*]$ which encodes the watermark number $w = 4$, and the dynamic call graph $G(P^*, I_{key})$ of the watermarked program P^*. We observe that the edge (f_4, f_6) in $G(P, I_{key})$ is not an edge in $F(\pi^*)$, and thus we correspond it in $G(P^*, I_{key})$ by a shortest path from f_i to f_j in $F(\pi^*)$, that is, the path (f_4, f_3, f_5, f_6).

Model Assessment. We evaluated our software watermarking model based on several criteria [8], that is model's performance and resilience. The performance criteria mainly focus on the data rate, the embedding overhead, the part protection and the credibility. Our results (see [7]) show that our watermarking model embeds a watermark without adding significant overhead compared to the initial program P. The resilience criteria mainly focus on the stealthiness and the distortion of the watermark. In order to evaluate these criteria we implemented our model on several application programs and tested its resilience under several attacks. Our results (see [6]) show that the watermarked program resists on decompilation, optimization and in most categories of obfuscation attacks as well as in programming language transformation.

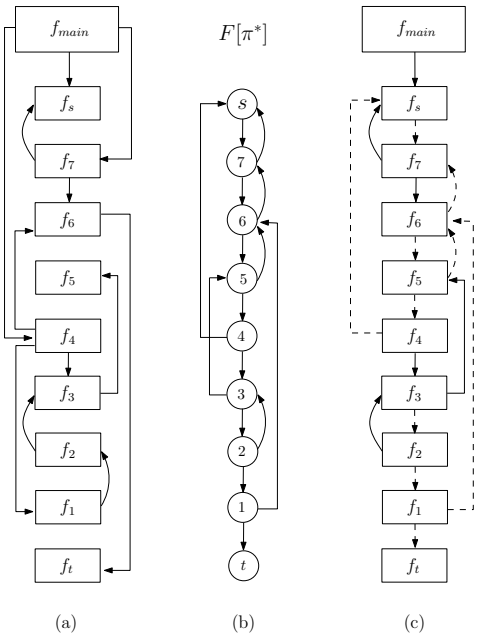

Fig. 2. (a) The dynamic call-graph $G(P, I_{key})$ of an application program P. (b) The reducible permutation graph $F[\pi^*]$. (c) The dynamic call-graph $G(P^*, I_{key})$ of the watermarked program P^*.

IV. Image Watermarking

We have developed watermarking techniques which make use of the structures described in Section II. In our image watermarking method we embed a watermark integer w into an image I by transforming it into a SiP, resulting in the watermarked image I_w. We represent a SiP in 2D space since images are 2D structures and thus we can efficiently embed a watermark by marking the high frequency bands of specific areas of the image I. The areas on the image are defined by a grid of $n \times n$ size that we place on I and it is of the same size with the 2DM representation matrix A^* of the watermark w.

We use the information stored in the 2DM representation matrix A^* (i.e. marks) in order to modify the corresponding cells in the image I. The main idea of our method is that we modify DFT's magnitude of specific high frequencies such that the least possible information is added ensuring thus robustness and imperceptiveness. The additive information is relied on the relationship between two groups of values which are not groups of pixels but frequencies. What is important in our method is the fact that we can successfully extract the watermark w from image I_w even after image I_w has been

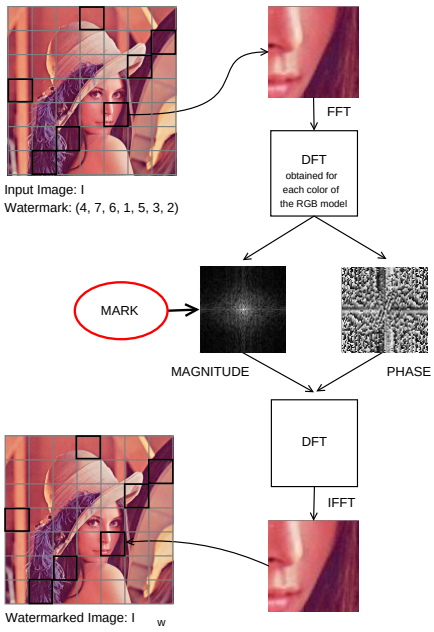

Fig. 3. A flow of the embedding process in image watermarking.

compressed with a lossy method. Figure 3 demonstrates the main operations performed by our embedding algorithm.

Model Assessment. We experimentally evaluated our embedding and extracting algorithms on digital color images of various sizes and quality characteristics based on two objective image quality assessment metrics namely Peak Signal to Noise Ratio (PSNR) and Structural Similarity Index Metric (SSIM). We obtained positive results since watermark embedding did not affect images' quality and the watermark w was extractable despite JPEG compression.

More precisely, after testing the algorithm for various images of different sizes and with different JPEG compression ratios, watermarks were extractable and the PSNR and SSIM values that we obtained remained over $40dB$ and over 0.9 respectively proving the algorithm's attribute of high fidelity. The pattern of the results was as follows. As image quality was dropping due to higher JPEG compression ratio the results from the quality metrics where dropping due to the more robust marks required and as images where getting smaller the quality metrics where dropping again as modifications in smaller images have greater impact. But as mentioned the results remained rather positive and the pattern was quite similar for different images of the same sizes.

What is more, thanks to certain properties of self-inverting permeations namely bionic properties [2], we are enabled to overcome geometric attacks, i.e., rotation and cropping, and recover the embedded watermarks without the loss or the distortion of information.

V. Audio Watermarking

Audio watermarking has received great attention since piracy in music industry leads to multibillion loss of profits and encryption methods cannot be used since the content must be played back in the original form. Digital audio watermarking, similarly with digital image and software watermarking, is a technique for protecting the intellectual property of the audio. The basic idea behind all audio watermarking schemes is that the watermark w is embedded in the host signal S producing thus the watermarked audio signal S_w.

Several techniques from signal processing, cryptography, coding theory, detection and estimation theory, information theory, and computer science have been proposed in the literature [12]. The idea of the proposed technique for audio watermarking comes from our image watermarking technique [4]. Since digital audio is an one-dimensional signal and a digital image is a two-dimensional signal, we apply in the 1D space a technique similar to image watermarking.

More precisely, in this work we propose an efficient algorithm for encoding a self-inverting permutation π^* into an audio signal S by first mapping the elements of π^* into a 1DM representation matrix B^* of size n^2 to mark specific time segments of audio signal S in the frequency domain resulting the watermarked audio signal S_w. We also propose an efficient algorithm for extracting the embedded self-inverting permutation π^* from the watermarked audio signal S_w by locating the positions of the marks in S_w.

A. Embed Watermark into Audio

We next describe the embedding algorithm of our proposed technique which encodes a self-inverting permutation (SiP) π^* into a digital audio signal S. Recall that the permutation π^* is obtained over the set N_{n^*}, where $n^* = 2n + 1$ and n is the length of the binary representation of an integer w which actually is the audio's watermark (authors' technique in [2]); see, Subsection II-A.

The watermark w, or equivalently the corresponding self-inverting permutation π^*, is invisible and it is inserted in the frequency domain of specific segments of the audio signal S. More precisely, we mark the high frequencies of DFT's magnitude for each audio segment using two rectangle areas, denoted hereafter as "Red" and "Blue". The rectangle areas are specified by the height and width as well as the distance from the mean of the DFT's magnitude matrix. In our implementation we use fixed widths and distance from the center of the DFT's magnitude matrix. The height is specified by the maximum value in the defined area.

The algorithm takes as input a SiP π^* and an audio signal S, in which the user embeds the watermark, and returns the watermarked audio signal S_w. We consider that our algorithm consists of four phases, where in the first phase we construct the 1DM representation of the watermark number w, in the second phase we transform the input signal and acquire the frequency representation of it while in the third phase we modify signals' frequencies according to the 1DM representation of the signal and finally the watermarked audio signal is returned (see Figure 4). We briefly describe below the four phases.

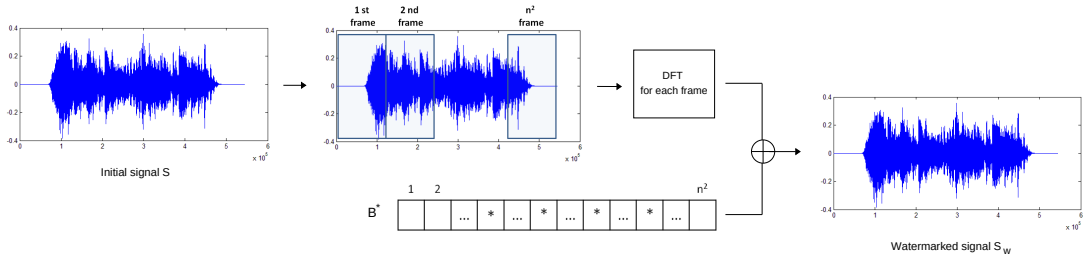

Fig. 4. The encoding process of audio signal watermarking.

Phase I. Compute the 1DM representation of the permutation π^*, i.e., construct a matrix B^* of size n^2.
Phase II. Segment the audio signal S in n^2 continuing and non overlapping frames f_i, $1 \leq i \leq n^2$, of size $\lfloor \frac{N-1}{n^2} \rfloor$, where N is the length of the audio signal. Then, for each frame f_i we apply the Discrete Fourier Transform (DFT) using the Fast Fourier Transform (FFT) algorithm and compute the magnitude matrices for each frame. In each magnitude matrix we define two areas of values, the "Red" and "Blue" which are symmetric to the center of the matrix.
Phase III. Assign each DFT magnitude matrix into a one-to-one correspondence with the elements of the matrix B^* and modify the DFT magnitude matrices that correspond to the marked cells of the matrix B^*. In fact, in the DFT magnitude matrices we modify the values of the "Red" rectangle area such that the averege values of the "Red" rectangle surpasses the average value of the "Blue" one by an appropriate factor.
Phase IV. Reconstruct the DFT of the corresponding modified magnitude matrices in order to obtain the watermarked audio signal S_w.
The "Red" and "Blue" Rectangle Areas. Concerning the two imaginary rectangle areas referred in this section, they are found in a frame's DFT magnitude matrix and their positions are selected in a similar manner to our image watermarking method in [4] where the two areas were found in the 2D space and they had the form of two ellipsoidal annuli with equal width $p = 2$ on the outer bounds of a magnitude cell. Now, in the audio case, we select two continuous rectangle areas in the 1D space on the left and right boundaries of a frame's magnitude matrix, keeping once again at each one a width equal to $p > 0$ and keeping symmetry in accordance to the center of the matrix. Similar to the annuli at the image watermarking technique, coming from the boundaries of the left side, we first come across the "Red" and then the "Blue" rectangle and from the right side it is the other way round.

B. Extract Watermark from Audio

The extracting algorithm of our proposed technique follows the reverse procedure of the embedding. The main idea is that the self-inverting permutation π^* is obtained from the frequency domain of specific frames of the watermarked audio signal S_w. More precisely, using the same two "Red" and "Blue" rectangles, we detect certain areas of the watermarked audio signal S_w that the average values of the "Red" rectangle exceed the average values of the "Blue" rectangle and these marked frames enable us to obtain the 1DM representation of the permutation π^*.

VI. Concluding Remarks

In this paper we survey algorithmic techniques for software and image watermarking and present a new developing idea based on them for audio watermarking. The key idea of all proposed techniques is based on three alternative representations of self-inverting permutations that we suggest for hiding information in each case. We leave as an open problem a thorough evaluation of our audio codec algorithms.

References

[1] L.M.S. Bento, D. Boccardo, R.C.S. Machado, V.G. Pereira de Sa, and J.L. Szwarcfiter, "Towards a provably resilient scheme for graph-based watermarking," 39th Int'l Workshop on Graph-Theoretic Concepts in Computer Science (WG'13), LNCS Proceedings, 2013.

[2] M. Chroni and S.D. Nikolopoulos, "Encoding watermark integers as self-inverting permutations," 11th Int'l Conference on Computer Systems and Technologies (CompSysTech'10), ACM ICPS 471, pp. 125–130, 2010.

[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, pp. 595–600, 2012.

[4] M. Chroni, A. Fylakis, and S.D. Nikolopoulos, "Watermarking Images using 2D Representations of Self-inverting Permutations," 8th Int'l Conference on Web Information Systems and Technologies (WEBIST'12), SciTePress Digital Library, pp. 380–385, 2012.

[5] M. Chroni and S.D. Nikolopoulos, "Design and Evaluation of a Graph Codec System for Software Watermarking," 2nd Int'l Conference on Data Management Technologies and Applications (DATA'13), SciTePress Digital Library, 2013.

[6] I. Chionis, M. Chroni, and S.D. Nikolopoulos, "A dynamic watermarking model for embedding reducible permutation graphs into software," 10th Int'l Conference on Security and Cryptography (SECRYPT'13), SciTePress Digital Library, 2013.

[7] I. Chionis, M. Chroni, and S.D. Nikolopoulos, "Evaluating the WaterRpg software watermarking model on Java application programs," 17th Panhellenic Conference on Informatics (PCI'13), ACM ICPS Proceegings, 2013.

[8] C. Collberg, A. Huntwork, E. Carter, G. Townsend, and M. Stepp, "More on graph theoretic software watermarks: Implementation, analysis, and attacks, Journal of Information and Software Technology," vol. 51, pp. 56–67, 2009.

[9] C. Collberg and J. Nagra, *Surreptitious Software*, Addison-Wesley, 2010.

[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.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press Inc., New York, 1980.

[12] F. Husain, "A Survey of Digital Watermarking Techniques for Multimedia Data," Int'l Journal of Electronics and Communication Engineering, vol. 2, No. 1, pp. 37–43, 2012.

[13] R. Sedgewick and P. Flajolet, *An Introduction to the Analysis of Algorithms*, Addison-Wesley, 1996.

[14] A. Tirkel, G. Rankin, R. van Schyndel, W. Ho, N. Mee, and C. Osborne, "Electronic water mark," Digital Image Computing: Technqiues and Applications (DICTA'93), pp. 666-672, 1993.