WATERMARKING IMAGES USING 2D REPRESENTATIONS OF SELF-INVERTING PERMUTATIONS

Maria Chroni, Angelos Fylakis and Stavros D. Nikolopoulos
Department of Computer Science, University of Ioannina, GR-45110 Ioannina, Greece

Keywords: Watermarking Techniques, Image Watermarking Systems, Intellectual Property Rights, Color Images, Self-inverting Permutations, 2D-representations of Permutations, Encoding, Decoding, Algorithms.

Abstract: In this work we propose an efficient and easily implemented codec system, which we named WaterIMAGE, for watermarking images that are intended for uploading on the web and making them public online. An important fact of our system is that it suggests a way in which an integer number can be represented in a two dimensional grid and, thus, since images are two dimensional objects that representation can be efficiently marked on them. In particular, our system uses an efficient technique which is based on a 2D representation of self-inverting permutations and mainly consists of two components: the first component contains an encoding algorithm which encodes an integer w into a self-inverting permutation π^* and a decoding algorithm which extracts the integer w from π^*, while the second component contains codec algorithms which are responsible for embedding a watermark into an image I, resulting the image I_w, and extract it from I_w. Our system incorporates important properties which allow us to successfully extract the watermark w from the image I_w even if the input image has been compressed with a lossy method and/or rotated. All the system's algorithms have been developed and tested in JAVA programming environment.

1 INTRODUCTION

Internet technology, becomes day by day an indispensable tool for everyday life since most people use it on a regular basis and do many daily activities online (Garfinkel, 2001). As a consequence, transferring digital information via the Internet, such as audio, pictures, video, or software, has also become very popular during the last years.This frequent use of internet means that measures taken for internet security are indispensable 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.And that's where watermarks come to place.

Watermarking. Watermarks are symbols which are placed into physical objects such as documents, photos and bank notes 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.

The watermarking problem can be described as the problem of embedding a watermark w into an object I and, thus, producing a new object I_w, such that w can be reliably located and extracted from I_w even after I_w has been subjected to transformations (Collberg and Nagra, 2010); for example, compression in case the object is an image.

Motivation. We believe that protecting intellectual material on the web is one of the major issues concerning the proper use of the internet. Digital images are a very characteristic part of this material found online and our target is to make people feel free to upload their images without hesitating because of the fear of their work being unauthorized used.

Watermarking is the ideal solution for protecting your property of the images and keeping them visible to the public as well, so research towards imperceptible secure and robust image watermarking techniques is vital. As mentioned, there are already various methods that can achieve that, but every single method requires attention and that's because we can not discriminate a specific method as the best. Every case has its ideal solution and the same idea applies for image watermarking.

Contribution. In this work we present an efficient and easily implemented codec system, which we named *WaterIMAGE*, for watermarking images that we are interested in uploading in the web and mak-

380 Chroni M., Fylakis A. and D. Nikolopoulos S..
WATERMARKING IMAGES USING 2D REPRESENTATIONS OF SELF-INVERTING PERMUTATIONS.
DOI: 10.5220/0003936003800385
In *Proceedings of the 8th International Conference on Web Information Systems and Technologies* (WEBIST-2012), pages 380-385
ISBN: 978-989-8565-08-2
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

ing them public online; this way web users are now enabled to consider how to protect their own images.

2 BASIC TOOLS

In this section we present basic tools which are used by our image watermarking system. In particular, we first describe discrete structures, namely, permutations, self-inverting permutations, and bitonic permutations, and then briefly outline a codec system (encoding/decoding algorithms) which encodes an integer w into a self-inverting permutation π^* and extracts it from π^*. Finally, we propose a 2D representation of permutations and give a 3D representation of color images.

2.1 Self-inverting Permutations

Informally, a permutation of a set of objects S is an arrangement of those objects into a particular order, while in a formal (mathematical) way a permutation of a set of objects S is defined as a bijection from S to itself (i.e., a map $S \rightarrow S$ for which every element of S occurs exactly once as image value)(Sedgewick and Flajolet, 1996; Golumbic, 1980).

Hereafter, we shall say that π^* is a permutation over the set N_9.

Definition 2.1.1. Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ be a permutation over the set N_n, where $n > 1$. The inverse of the permutation π is the permutation $\tau = (q_1, q_2, \ldots, q_n)$ 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, every permutation has a unique inverse, and the inverse of the inverse is the original permutation. Clearly, a permutation is a SIP (self-inverting permutation) if and only if all its cycles are of length 1 or 2; hereafter, we shall denote a 2-cycle as $c = (x,y)$ and an 1-cycle as $c = (x)$, or, equivalently, $c = (x,x)$.

The permutation $\pi^* = (5,6,9,8,1,2,7,4,3)$ is a SIP with cycles: $(1,5)$, $(2,6)$, $(3,9)$, $(4,8)$, and $(7,7)$.

2.2 Encoding Numbers as SIPs

Next, we present an algorithm for encoding an integer w into a self-inverting permutation π^* and an algorithm for extracting w from π^*; both algorithms run in $O(n)$ time, where n is the length of the binary representation of the integer w (author's algorithms). The encoding process uses the notion of *Bitonic Permutations* which we briefly describe below.

Bitonic Permutations. The key-object in our algorithm for encoding integers as self-inverting permutations is the bitonic permutation: a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ over the set N_n is called bitonic if either monotonically increases and then monotonically decreases, or else monotonically decreases and then monotonically increases. For example, the permutations $\pi_1 = (1,4,6,7,5,3,2)$ and $\pi_2 = (6,4,3,1,2,5,7)$ are both bitonic.

Our encoding algorithm uses only bitonic permutations that monotonically increase and then monotonically decrease. Let π be such a bitonic permutation over the set N_n and let π_i, π_{i+1} be the two consecutive elements of π such that $\pi_i > \pi_{i+1}$. Then, the sequence $X = (\pi_1, \pi_2, \ldots, \pi_i)$ is called first increasing subsequence of π and the sequence $Y = (\pi_{i+1}, \pi_{i+2}, \ldots, \pi_n)$ is called first decreasing subsequence of π.

We next give some notations and terminology we shall use throughout the encoding and decoding process. Let w be an integer number. We denote by $B = b_1 b_2 \cdots b_n$ the binary representation of w. If $B_1 = b_1 b_2 \cdots b_n$ and $B_2 = d_1 d_2 \cdots d_m$ be two binary numbers, then the number $B_1 || B_2$ is the binary number $b_1 b_2 \cdots b_n d_1 d_2 \cdots d_m$. The binary sequence of the number $B = b_1 b_2 \cdots b_n$ is the sequence $B^* = (b_1, b_2, \ldots, b_n)$ of length n.

Let $B = b_1 b_2 \cdots bn$ be a binary number. Then, $flip(B) = b'_1 b'_2 \cdots b'_n$ is the binary number such that $b'_i = 0$ (1 resp.) if and only if $b_i = 1$ (0 resp.), $1 \leq i \leq n$.

Algorithm W-to-SIP: We briefly outline an algorithm for encoding an integer as self-inverting permutation. Our algorithm, which we call Encode_W-to-SIP, takes as input an integer w, computes the binary representation $b_1 b_2 \cdots b_n$ of w, and then produces a self-inverting permutation π^* in $O(n)$ time (Chroni and Nikolopoulos, 2010).

Algorithm SIP-to-W: Having presented the encoding algorithm Encode_W-to-SIP, let us now present an extraction algorithm, that is, an algorithm for decoding a self-inverting permutation. More precisely, our extraction algorithm, which we call Decode_SIP-to-W, takes as input a self-inverting permutation π^* produced by the algorithm Encode_W-to-SIP and returns its corresponding integer w. The time complexity of the decode algorithm is also $O(n)$, where n is the length of the permutation π^* (Chroni and Nikolopoulos, 2010).

2.3 2DM Representations

Given a permutation π over the set $N_n = \{1, 2, \ldots, n\}$,

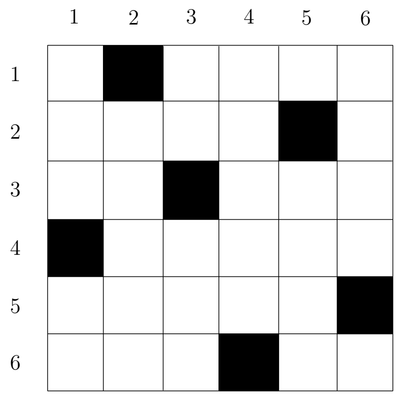

Figure 1: A 2DM representation of the self-inverting permutation $\pi = (5,6,9,8,1,2,7,4,3)$.

we first define a two-dimensional representation (2D-representation) of the permutation π that is useful for studying properties which help us to define, later, a more suitable representation of π for efficient use in our watermarking system.

In this representation, the elements of the permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ are mapped in specific cells of an $n \times n$ matrix A as follows:

- integer $i \longrightarrow$ entry $A(\pi_i^{-1}, i)$

or, equivalently,

- the cell at row i and column π_i is labeled by the number π_i, for each $i = 1, 2, \ldots, n$.

Based on the previous 2D representation of a permutation, we next propose a two-dimensional marked representation (2DM representation) of a permutation which is an efficient tool for watermarking images.

In our 2DM representation, a permutation π over the set $N_n = \{1, 2, \ldots, n\}$ is represented by an $n \times n$ matrix A^* as follows:

- the cell at row i and column π_i is marked by a specific symbol, for each $i = 1, 2, \ldots, n$.

Figure 1 shows the 2DM representation of the permutation π. Note that, as in the 2D representation, there is also one symbol in each row and in each column of the matrix A^*.

We next present an algorithm which extracts the permutation π from its 2DM representation matrix. More precisely, let π be a permutation over N_n and let A^* be the 2DM representation matrix of π (see, Figure 1); given the matrix A^*, we can easily extract π from A^* in linear time (in the size of matrix A^*) by the following algorithm:

```
Algorithm Extract_π_from_2DM
Input: the 2DM representation matrix $A^*$ of $\pi$;
Output: the permutation $\pi$;

1. For each row $i$ of matrix $A^*$, $1 \le i \le n$, do:
       find the marked cell and let $j$ be its column;
       set $\pi_i \leftarrow j$;
2. Return the permutation $\pi$;
```

Remark 2.3.1. It is easy to see that the resulting permutation π, after the execution of Step 1, can be taken by reading the matrix A^* from top row to bottom row and write down the positions of its marked cells. Since the permutation π is a self-inverting permutation, its 2D matrix A has the following property:

- $A(i,j) = j$ if $\pi_i = j$, and
- $A(i,j) = 0$ otherwise, $1 \le i, j \le n$.

Thus, the corresponding matrix A^* is symmetric:

- $A^*(i,j) = A^*(j,i) =$ "mark" if $\pi_i = j$, and
- $A^*(i,j) = A^*(j,i) = 0$ otherwise, $1 \le i, j \le n$.

Based on this property, it is also easy to see that the resulting permutation π can be also taken by reading the matrix A^* from left column to right column and write down the positions of its marked cells.

2.4 Color Images

A digital image is a numeric representation of a 2-dimensional image; it has a finite set of values, called *picture elements* or *pixels*, that represent the brightness of a given color at any specific point in the image.

In our system we use the *RGB* model, where the name comes from the initials of the three additive colors Red, Green and Blue. The range of colors can be represented on the Cartesian 3-dimensional system. The axes x, y and z are used for the red green and blue color respectively

In our system, since a color is a triple of integers (x,y,z), a digital image I of resolution $N \times M$ (i.e., it contains N rows and M columns of pixels) is stored in a three-dimensional matrix *Img* of size $N \times M \times 3$ as follows:

if the pixel $I(i,j)$ of the image I has (x,y,z) color, then $Img(i,j,1) = x$, $Img(i,j,2) = y$, and $Img(i,j,3) = z$.

3 OUR IMAGE WATERMARKING SYSTEM

Having proposed an efficient method for encoding integers as self-inverting permutations using the bitonic property of a permutation, and the 2DM representation of self-inverting permutations, we next describe

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

the two main algorithms of our image watermarking system; the encoding algorithm Encode_SIP-to-IMAGE which encodes a self-inverting permutation π^*, corresponding to watermark w, into an image I resulting the watermarked image I_w and the decoding algorithm Decode_IMAGE-to-SIP which extracts the permutation π^* from the image I_w.

3.1 Embed Watermark into Image

We next describe the algorithm Encode_SIP-to-IMAGE of our codec system which embeds a self-inverting permutation (SIP) π^* into an image I; recall that, in our system we use a SIP π^* over the set N_{n^*} for encoding the watermark w, where $n^* = 2n+1$ and n is the length of the binary representation of the integer w (author's technique); see, Subsection 2.2.

The algorithm takes as input a SIP π^* and an image I, in which the user wants to embed the watermark $w = \pi^*$, and produces the watermarked image I_w; it works as follows:

Step 1: The algorithm first computes the 2DM representation of the permutation π^*, that is, it computes the $n^* \times n^*$ array A (see, Subsection 2.3); the entry (i, π_i^*) of the array A contains the symbol "*", $1 \le i \le n^*$.

Step 2: Next, the algorithm computes the size $N \times M$ of the input image I and do the following: if N is an even number it removes the pixels from the bottom row of I and reduces N by 1, while if M is an even number it removes the pixels from the right column of I and reduces M by 1. The resulting image has size $N^* \times M^*$, where N^* and M^* are both odd numbers.

Step 3: Let n^* be the size of the SIP π^* and let $N^* \le M^*$. Now the algorithm takes the input image I and places on it an imaginary grid $\mathcal{G}$, which covers almost the whole image I, having

$$n^* \times n^* \text{ grid-cells } C_{ij}(I)$$

each $C_{ij}(I)$ of size

$$\lfloor N^*/n^* \rfloor \times \lfloor N^*/n^* \rfloor$$

where, $1 \le i, j \le n^*$.

It places the imaginary grid $\mathcal{G}$ on I as follows: it first locates the central pixel P^0_{cent} of the image I, which is at position $(\lfloor N^*/2 \rfloor + 1, \lfloor M^*/2 \rfloor + 1)$, then locates the central pixel p^0_{ii} of the central grid-cell $C_{ii}(I)$, where $i = \lfloor n^*/2 \rfloor + 1$, and places the grid $\mathcal{G}$ on image I such that both P^0_{cent} and p^0_{ii} have the same position in I.

Step 4: Then it scans the image and goes to each grid-cell $C_{ij}(I)$ (there are always $n^* \times n^*$ grid-cells in any image) and locates the central pixel p^0_{ij} of the grid-cell $C_{ij}(I)$ and also the four pixels p^1_{ij}, p^2_{ij}, p^3_{ij}, and p^4_{ij} around it, $1 \le i, j \le n^*$; hereafter, we shall call these four pixels *cross* pixels.

Then, it computes the difference between the brightness of the central pixel p^0_{ij} and the average brightness of the twelve pixels around it, that is, the pixels $p^{\ell 1}_{ij}$, $p^{\ell 2}_{ij}$, and $p^{\ell 3}_{ij}$ ($\ell = 1, 2, 3, 4$), and stores this value in the variable $\mathrm{dif}(p^0_{ij})$ (see, Figure 2).

Finally, it computes the maximum absolute value of all $n^* \times n^*$ differences $\mathrm{dif}(p^0_{ij})$, $1 \le i, j \le n^*$, and stores it in the variable $\mathrm{Maxdif}(I)$.

Step 5: The algorithm goes again to each central pixel p^0_{ij} of each grid-cell C_{ij} and if the corresponding entry $A(i, j)$ contains the symbol "*", then it increases

- the brightness k^0_{ij} of the central pixel p^0_{ij}, and
- the brightness k^1_{ij}, k^2_{ij}, k^3_{ij}, and k^4_{ij} of its cross pixels.

Actually, it first increases the central pixel p^0_{ij} by the value e^0_{ij} so that it surpasses the image's maximum difference $\mathrm{Maxdif}(I)$ by a constant c; that is,

- $k^0_{ij} + e^0_{ij} = \mathrm{Maxdif}(I) + c$

and, then, it sets the brightness of the four cross pixels p^1_{ij}, p^2_{ij}, p^3_{ij}, and p^4_{ij} equal to k^0_{ij}.

In our system we use $c = 5$, and thus the brightness k^0_{ij} of the central pixel of each grid-cell C_{ij} is increased by e^0_{ij}, where

$$e^0_{ij} = \mathrm{Maxdif}(I) - k^0_{ij} + 5 \qquad (1)$$

where, $1 \le i, j \le n^*$.

Let I_w be the resulting image after increasing the brightness of the n^* central and the corresponding cross pixels, with respect to π, of the image I. Hereafter, we call the n^* central pixels of I as *2DM-pixels*; recall that, p^0_{ij} is a 2DM-pixel if $A(i, \pi_i)$ = "*", or, equivalently, the cell (i, π_i) of the matrix A is marked.

Step 6: The algorithm returns the watermarked image I_w.

3.2 Extract Watermark from Image

Next we describe our decoding algorithm which is responsible for extracting the watermark $w = \pi^*$ form image I_w. In particular, the algorithm, which we call Decode_IMAGE-to-SIP, takes as input a watermarked image I_w and returns the SIP π^* which corresponds to integer watermark w; the steps of the algorithm are the following:

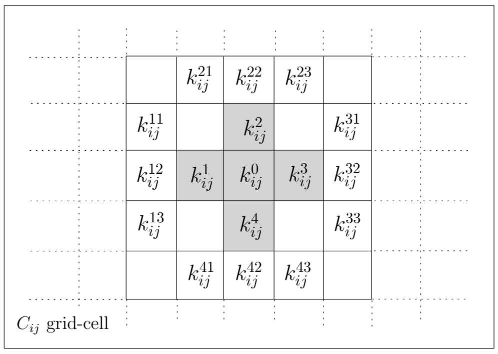

Figure 2: The brightness k_{ij}^{ℓ} of the central and cross pixels p_{ij}^{ℓ} of the grid-cell $C_{ij}(I)$, $0 \leq \ell \leq 4$, and the brightness $k_{ij}^{\ell m}$ of the cycle-cross pixels $p_{ij}^{\ell m}$, $1 \leq \ell \leq 4$ and $m = 1,2,3$.

Step 1: The algorithm places again the same imaginary $n^* \times n^*$ grid on image I_w and locates the central pixel p_{ij}^0 of each grid-cell $C_{ij}(I)$, $1 \leq i, j \leq n^*$; there are $n^* \times n^*$ central pixels in total. Then, it finds the n^* central pixels $p_1^0, p_2^0, \ldots, p_{n^*}^0$, among the $n^* \times n^*$, with the maximum brightness using a known sorting algorithm (Cormen et al., 2001).

Step 2: In this step, the algorithm takes the n^* grid-cell $C_1, C_2, \ldots, C_{n^*}$ of the image I_w which correspond to n^* central pixels $p_1^0, p_2^0, \ldots, p_{n^*}^0$, and compute an $n^* \times n^*$ matrix A^* as follows:

- Initially, set $A^*(i,j) \leftarrow 0$, $1 \leq i, j \leq n^*$;
- For each grid-cell C_m, $1 \leq m \leq n^*$, do:

  if (i,j) is the position of the grid-cell C_m in the grid $\mathcal{G}$ then set $A^*(i,j) \leftarrow$ "$*$";

It is easy to see that, the $n^* \times n^*$ matrix A^* is exactly the 2DM representation of the self-inverting permutation π^* embedded in image I_w by the algorithm Encode_SIP-to-IMAGE.

Then, the permutation π^* can be extracted from the matrix A^* using the algorithm Extract_π_from_2DM; see, Subsection 2.3.

Step 3: Finally, the algorithm returns the self-inverting permutation π^*.

4 PERFORMANCE

Our image watermarking system mainly consists of four algorithms, each of which is responsible for a particular codec operation:

- Encode_W-to-SIP: algorithm for encoding an integer w into a self-inverting permutation π^*;
- Decode_SIP-to-W: algorithm for extracting w from π^*;
- Encode_SIP-to-IMAGE: algorithm for encoding a self-inverting permutation π^* into an integer I;
- Decode_IMAGE-to-SIP: algorithm for extracting π^* form the watermarked image;

The two algorithms that are considered the basic ones for our system are those responsible for embedding a SIP into an image and extracting the SIP from it.

We next discuss some issues concerning the performance of our image watermarking system. In particular, we mainly focus on the embedding algorithm Encode_SIP-to-IMAGE and the efficiency of watermarking image I_w produced by this algorithm, and also on important properties of the $n^* \times n^*$ matrix A^* which stores the 2DM representation of a SIP. Finally we show the time and space performance of our system by computing the complexity of their algorithms.

It is worth noting that our system incorporates such properties which allow us to successfully extract the watermark w from the image I_w even if the input image I_w of algorithm Decode_IMAGE-to-SIP has been compressed with a lossy method and/or rotated.

We have evaluated the embedding and extracting algorithms by testing them on various and different in characteristics images that were initially in JPEG format and we had positive results as the watermark was successfully extracted at every case even if the image was converted back into JPEG format. What is more, the method is open to extensions as the same method might be used with a different marking procedure part of the Encode_SIP-to-IMAGE algorithm.

All the system's algorithms have been initially developed and tested in MATLAB (Gonzalez et al., 2003) and then redeveloped and also tested in JAVA.

Compression. The experimental results show that the watermark w is "well hidden" in the image I_w. We believe that it is because we mark the image by changing the difference between the brightness of the 2DM-pixels p_{ij}^0 of the $n^* \times n^*$ imaginary grid and its 12 neighborhood pixels around it, that is, the pixels $p_{ij}^{\ell 1}$, $p_{ij}^{\ell 2}$, and $p_{ij}^{\ell 3}$, for $\ell = 1,2,3,4$ (see, Figure 2 and also Step 2 of the embedding algorithm Encode_SIP-to-IMAGE); recall that, we also set the brightness of the four cross pixels of each 2DM-pixel p_{ij}^0, that is, the pixels p_{ij}^1, p_{ij}^2, p_{ij}^3, and p_{ij}^4, to be equal to the brightness of the 2DM-pixel p_{ij}^0.

Note that, we change the brightness of the 2DM-pixels by increasing them so that they surpass the image's maximum difference Maxdif(I) by a constant c, where in our implementation $c = 5$. We add five because if we compress the image the values of the pixels may slightly change, and we want our watermark to be robust. We also believe that this technique despite being simple, it is efficient because the brightness of each of the n^* marked central pixels does not have a great difference from the brightness of the 12 neighborhood pixels and thus the modified central pixel, along with the cross pixels, does not change something significantly in the image.

Rotation. The watermarked images produced by our embedding method have a property worth to be referenced. And this is certain characteristics noticed at the 2DM representation of the image's watermarks which in our system are self-inverting permutations. Sometimes an image might show an indeterminate depiction, such as a night sky or an aerial view. These types of images might be rotated changing the coordinates of the watermark's marks making invalid the watermark that we are about to extract. Also it is about an indeterminate depiction which does not allow someone to tell which is the right angle of the image.

Thanks to our embedding method's properties this problem can be overcome. It has to do with the coordinates of the marks of a 2DM representation of a self-inverting permutation found on image I_w. Those coordinates allow us to turn the image into the initial angle and then extract the watermark successfully.

The 2DM representation of a self-inverting permutation has the following properties:

(i) The main diagonal of the $n^* \times n^*$ symmetric matrix A^* have always one and only one marked cell, and

(ii) this marked cell are always in the entries (i,i) of A^*, where $i = \lceil \frac{n^*}{2} \rceil + 1, \lceil \frac{n^*}{2} \rceil + 2, \ldots, n^*$.

If the main diagonal of matrix A^* has no marked cell then we rotate the image by 90 degrees. Additionally, if the marked cell of the main diagonal is in entry (i,i) with $i \leq \lceil \frac{n^*}{2} \rceil$, then we turn the image by 180 degrees and thus we end up at the initial image from which we are able to extract the right watermark.

Time and Space Complexity. The total time performance of our codec system, neglecting the conversion of the input image I into raw raster format, is $N \times n^*$ for embedding the watermark w into I and $N + M + (n^* \times n^*) \times \log(n^* \times n^*)$ for extracting w from the watermarked image I_w. Moreover, the extra space needed by our codec system is linear in the size of the input image, i.e., it uses only some extra auxiliary variables and an auxiliary matrix for the 2DM representation of the self-inverting permutation.

5 CONCLUDING REMARKS

In this paper, we proposed a codec system, which we named WaterIMAGE, for watermarking images that are intended for online publication.

The proposed WaterIMAGE system has the following design and functional advantages:

- it is an efficient image watermarking system; the experimental results showed that the watermark w is "well hidden" in the image I_w,
- its embedding method incorporates properties that allow us to successfully extract the watermark w for the image I_w even if the image I_w has been compressed with a lossy method and/or rotated,
- it is a simple and easily implemented system, and
- finally, as far as the time and space needed for the encoding/decoding process, it performs very well.

We should point out that the main feature of our WaterIMAGE system is the fact that it uses a combinatorial object to watermark an image; we show that an integer can be efficiently represented by a self-inverting permutation which, in turn, can be represented in the 2-dimensional space and, thus, this representation forms a suitable watermarking object for images. In our system we propose a marking method but apart from that the investigation of alternative and more efficient methods for marking an image using a 2D representation of a permutation are an open problem for further research.

REFERENCES

Chroni, M. and Nikolopoulos, S. (2010). Encoding watermark integers as self-inverting permutations. In *Proc. Int'l Conference on Computer Systems and Technologies (CompSysTech'10)*, volume ACM ICPS 471, pages 125 – 130.

Collberg, C. and Nagra, J. (2010). *Surreptitious Software*. Addison-Wesley.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press, 2nd edition.

Garfinkel, S. (2001). *Web Security, Privacy and Commerce*. O'Reilly, 2nd edition.

Golumbic, M. (1980). *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, Inc., New York.

Gonzalez, R. C., Woods, R. E., and Eddins, S. L. (2003). *Digital Image Processing using Matlab*. Prentice-Hall.

Sedgewick, R. and Flajolet, P. (1996). *An Introduction to the Analysis of Algorithms*. Addison-Wesley.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS