Reversible Digital Watermarking using Integer Wavelet Transform

Sambaran Bandyopadhyay
Department of Computer Science and Engineering
Institute of Engineering and Management
Kolkata, India–700091
Email: sam_krish89@rediffmail.com

Ruchira Naskar and Rajat Subhra Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Kharagpur, India–721302
Email: {ruchira,rschakraborty}@cse.iitkgp.ernet.in

***Abstract*—*Digital Watermarking* is a well–known class of techniques for digital content protection. Recently, *Reversible Digital Watermarking* techniques have drawn a lot of interest, where after the watermark has been extracted, the original content can be retrieved with zero distortion. In this paper we present a novel high capacity reversible digital watermarking technique based on integer wavelet transforms. Our simulation results fare reasonably well when compared to state–of–the–art reversible watermarking techniques of similar principle published in the literature.**

***Index Terms*—Reversible integer transform, reversible data embedding, PSNR, embedding capacity.**

I. Introduction

Digital watermarking is a class of popular techniques whereby hard–to–detect information (called the "signature" or "payload") is embedded in digital content (audio, image or video) for purposes of content authentication and intellectual property (IP) protection. Since only the creator or distributor of the digital content has knowledge about the hidden information and how to retrieve it, she can prove her ownership in case of litigation. In many application domains such as medical and military imaging, the original information is extremely sensitive and recovery of the original information in an unaltered form is of utmost importance. In such cases, *reversible watermarking* techniques have been found useful where by the very nature of the watermarking scheme, the original content can be retrieved exactly with zero distortion [1]–[3].

In this paper we present a high quality, high capacity reversible watermarking scheme for images. We use integer wavelet transform [1] to convert the original image into a set of average and difference numbers, and then repeat the same procedure for the reduced matrix. In our scheme each row of the original image matrix [4] is replaced by a single average number and multiple difference numbers. Since usually the difference numbers can be encoded in relatively fewer number of bits, through our technique we create space to embed larger number of payload bits in the difference numbers. The visual quality of the watermarked image compared to the original image is also found to be satisfactory, and is reflected in the calculated *peak signal-to-noise ratio* (PSNR).

The rest of the paper is organized as follows: in Section II we provide the mathematical formulation of integer wavelet transform based reversible watermarking. In Section III, we describe the methodology proposed in this paper. In Section IV, we present experimental results of applying the proposed algorithm to an example image. We conclude in Section V.

II. Background

A. Reversible Integer Wavelet Transform

The *integer wavelet transform* maps integers to integers, and allows for perfect invertibility with finite precision arithmetic. Also, the integer wavelet transform can be implemented with only three operations – addition, subtraction and shift, on a digital computer. This feature makes it attractive compared to other discrete wavelet transforms. For example, for the *Haar wavelet filter*, the integer wavelet transforms are:

$$l_i = \left\lfloor \frac{x_{2i} + x_{2i+1}}{2} \right\rfloor, h_i = x_{2i} - x_{2i+1} \tag{1}$$

where $\lfloor\rfloor$ implies the "floor function" which means the "greatest integer less than or equal to". The corresponding inverse transforms are:

$$x_{2i} = l_i + \left\lfloor \frac{h_i + 1}{2} \right\rfloor, x_{2i+1} = l_i - \left\lfloor \frac{h_i}{2} \right\rfloor \tag{2}$$

B. Watermarking based on Integer Wavelet Transform

Reversible watermarking is based on applying the above integer wavelet transform on the pixel encoded values, and utilizing the high spatial redundancy in pixel values in natural images. Let (x, y) be two pixel values in a grayscale image utilizing 8–bit binary encoding, where $x, y \in [0, 255]$. Then, the following values are computed:

$$l = \left\lfloor \frac{x + y}{2} \right\rfloor, h = x - y \tag{3}$$

Due to the high redundancy in natural images, the difference values h are usually comparatively smaller, and can be encoded using less than eight bits. The space saved can be thus utilized to embed the bits of the signature to be embedded. As an example, consider $x = 205$, $y = 200$, $l = 202$, $h = 5 = 101_2$. Suppose a bit $b = 0$ of information to be embedded at the location right after the most significant bit (MSB) in the binary representation of h. Then, the modified

value of h becomes $h^{'} = 1001_2 = 9$. Thus, the new grayscale values are:

$$x^{'} = l + \left\lfloor \frac{h^{'}+1}{2} \right\rfloor = 207, y^{'} = x^{'} - h^{'} = 198$$

From the embedded pair $(x^{'}, y^{'})$, the watermark detector can extract the embedded bit b and get back the original pair (x, y) by:

$$l^{'} = \left\lfloor \frac{x^{'}+y^{'}}{2} \right\rfloor = 202, h^{'} = x^{'} - y^{'} = 9 = 1001_2$$

Note that the values of l and $l^{'}$ are the same. With the knowledge of the location of the inserted watermark bit, the original difference value $h = 5 = 101_2$ can be extracted from $h^{'}$, and with the average number $l^{'}$ and the difference number h, the original values (x, y) can be re-calculated using the inverse integer transform. The above procedure of embedding the digital watermark by expanding the difference values is generally termed as *difference expansion*.

Difficulty arises when the value of h is large, which can lead to underflow or overflow conditions with the values to be embedded. For example, let $x = 105, y = 22$, then $l = 63, h = x - y = 83 = 1010011_2$. If we embed a bit "0" in h, the new value is $h^{'} = 10010011_2 = 147$. This leads to the embedded values $x^{'} = 137$ and $y^{'} = -10$. This will cause an underflow problem as grayscale values are restricted in the range $[0, 255]$. To restrict the overflow or underflow conditions, the following conditions must be satisfied:

$$0 \leq l + \left\lfloor \frac{h+1}{2} \right\rfloor \leq 255, 0 \leq l - \left\lfloor \frac{h}{2} \right\rfloor \leq 255$$

which is equivalent to

$$|h| \leq min(2(255 - l), 2l + 1) \tag{4}$$

The least significant bit (LSB) of the difference h is usually selected as the embedding area. Since

$$h = \left\lfloor \frac{h}{2} \right\rfloor \cdot 2 + LSB(h)$$

for $LSB(h) = 0$ or 1, the difference number h is *changeable* if

$$\left| \left\lfloor \frac{h}{2} \right\rfloor \cdot 2 + b \right| \leq min(2(255 - l), 2l + 1) \tag{5}$$

for both $b = 0$ and 1.

Note that modifying changeable h (without compression) does not provide additional storage space. The extra storage space is gained from *expandable* difference numbers. For a grayscale pixel pair (x, y), its difference number h is expandable if

$$|2 \cdot h + b| \leq min(2(255 - l), 2l + 1) \tag{6}$$

for both $b = 0$ and 1. For each expandable difference number, we get at least one bit of space to embed the watermark.

The information about the pixel locations and bit positions in the binary values of the pixels where the bits of the watermark are to be inserted is stored in a *location map*. The location map of the expanded difference numbers is usually in the form of a "bi-level image", where the pixel value is "1" at each location where it is expanded, and "0" otherwise. It is usually losslessly compressed using a compression technique such as JBIG2 or *run–length coding*. Similarly, the concatenation of the LSBs of the changeable difference numbers can be further compressed using *arithmetic coding* or *Huffman coding*. Optionally, a secure hash of the original digital image can be created by an algorithm such as SHA–256. All of the above bitstreams are then combined into a final bitstream and transmitted. In this work, for simplicity, we have not embedded the location map or hash information in the image, and have not applied any lossless compression algorithm to further compress these information.

III. Methodology

A. Multi-bit Hiding

Till now we have been concerned about embedding one extra bit of the watermark per difference number. But, we can go further by checking whether more than one bit can be embedded into a single difference number. We can check it with the help of the ***hiding ability*** of the difference number h. For a given difference number h, let k be the largest integer such that

$$|k \cdot h + b| \leq min(2(255 - l), 2l + 1) \tag{7}$$

for all $0 \leq b \leq k - 1$, where b is not necessarily a single bit any more. In such a case, we say that the hiding ability of h is $\log_2 k$. Hiding ability gives us the information of how many bits we can embed into the difference number without causing an underflow or overflow. Higher the value of hiding ability, better the embedding capacity. In most practical images, the hiding capacity of a difference term (calculated for two adjacent pixel values) is greater than one. For a difference number to be expandable, the value of hiding ability must be at least one (as $\log_2 2 = 1$).

Hiding ability also indirectly helps us to select the expandable difference numbers for data embedding. For each row of the reduced image matrix (which is in the form of a collection of average–difference value pairs), we examine the pairs of average values and replace them by another average value–difference value pair. We repeat this procedure for the reduced average–difference matrix to have multiple difference values (not necessarily expandable) and a single average value for each row. Since a large fraction of the set of difference values are expandable, a large number of bits of the payload can be embedded in them. Thus, in effect, the embedding capacity of the image can be improved which increases the signal–to–noise ratio for larger payload sizes. This observation is the main contribution of this work.

In the next section we describe the proposed watermark embedding and extraction for multi–bit hiding.

Algorithm 1 Procedure *EMBED_WATERMARK*

Embed the given watermark payload in a given grayscale image

Inputs: An image matrix X $(m \times n)$, watermark payload vector msg $(1 \times p)$ to be embedded

Outputs: Watermarked image matrix X $(m \times n)$, location map vector loc $(1 \times 2p)$

1: /* Perform integer wavelet transform and embed payload*/
2: Initialize each element of $avgdiff$ to zero
3: $t \leftarrow \log_2 n$ /*Assuming n to be a power of 2*/
4: $le \leftarrow 1$
5: **for** $t1 = 0$ to $(t-1)$ **do**
6: **for** $i = 1$ to m **do**
7: $j \leftarrow 1$
8: $k \leftarrow 2^{t1} + 1$
9: **while** $j \leq n$ and $k \leq n$ **do**
10: $avgdiff(i,j) \leftarrow \left\lfloor \frac{X(i,j)+X(i,k)}{2} \right\rfloor$
11: $avgdiff(i,k) \leftarrow X(i,j) - X(i,k)$
12: **if** $avgdiff(i,k)$ is expandable under the integer value $avgdiff(i,j)$ and $le \leq p$ **then**
13: $avgdiff(i,k) \leftarrow 2 \cdot avgdiff(i,k) + msg(le)$
14: Enter the location of the term $avgdiff(i,k)$ into the location map loc
15: $le \leftarrow le + 1$
16: **end if**
17: $j \leftarrow j + 2^{(t1+1)}$
18: $k \leftarrow k + 2^{(t1+1)}$
19: **end while**
20: **end for**
21: $X \leftarrow avgdiff$
22: **end for**
23: /*Calculate pixel values of the watermarked image*/
24: **for** $t1 = (t-1)$ downto 0 **do**
25: **for** $i = 1$ to m **do**
26: $j \leftarrow 1$
27: $k \leftarrow 2^{t1} + 1$
28: **while** $j \leq n$ and $k \leq n$ **do**
29: $X(i,j) \leftarrow avgdiff(i,j) + \left\lfloor \frac{avgdiff(i,k)+1}{2} \right\rfloor$
30: $X(i,k) \leftarrow avgdiff(i,j) - \left\lfloor \frac{avgdiff(i,k)}{2} \right\rfloor$
31: $j \leftarrow j + 2^{(t1+1)}$
32: $k \leftarrow k + 2^{(t1+1)}$
33: **end while**
34: **end for**
35: $avgdiff \leftarrow X$
36: **end for**

Algorithm 2 Procedure *EXTRACT_WATERMARK*

Extract the given watermark payload in a given grayscale image

Inputs: Watermarked image matrix X $(m \times n)$, location map vector loc $(1 \times 2p)$

Outputs: Original image matrix X $(m \times n)$, watermark payload msg $(1 \times p)$

1: Set $avgdiff \leftarrow (m \times n)$ null matrix
2: /* Extract payload from watermarked image*/
3: $t \leftarrow \log_2 n$ /*Assuming n to be a power of 2*/
4: **for** $t1 = 0$ to $(t-1)$ **do**
5: **for** $i = 1$ to m **do**
6: $j \leftarrow 1$
7: $k \leftarrow 2^{t1} + 1$
8: **while** $j \leq n$ and $k \leq n$ **do**
9: $avgdiff(i,j) \leftarrow \left\lfloor \frac{X(i,j)+X(i,k)}{2} \right\rfloor$
10: $avgdiff(i,k) \leftarrow X(i,j) - X(i,k)$
11: $j \leftarrow j + 2^{(t1+1)}$
12: $k \leftarrow k + 2^{(t1+1)}$
13: **end while**
14: **end for**
15: $X \leftarrow avgdiff$
16: **end for**
17: $k \leftarrow 1$
18: **while** $k \leq length(loc)$ **do**
19: $i \leftarrow loc(k)$
20: $j \leftarrow loc(k+1)$
21: $msg\left(\frac{k+1}{2}\right) \leftarrow X(i,j)\%2$ /*Get LSB*/
22: $X(i,j) \leftarrow \left\lfloor \frac{X(i,j)}{2} \right\rfloor$
23: $k \leftarrow (k+2)$
24: **end while**
25: /*Retrieve pixel values of the original image*/
26: $avgdiff \leftarrow X$
27: **for** $t1 = (t-1)$ downto 0 **do**
28: **for** $i = 1$ to m **do**
29: $j \leftarrow 1$
30: $k \leftarrow 2^{t1} + 1$
31: **while** $j \leq n$ and $k \leq n$ **do**
32: $X(i,j) \leftarrow avgdiff(i,j) + \left\lfloor \frac{avgdiff(i,k)+1}{2} \right\rfloor$
33: $X(i,k) \leftarrow avgdiff(i,j) - \left\lfloor \frac{avgdiff(i,k)}{2} \right\rfloor$
34: $j \leftarrow j + 2^{(t1+1)}$
35: $k \leftarrow k + 2^{(t1+1)}$
36: **end while**
37: **end for**
38: $avgdiff \leftarrow X$
39: **end for**

B. Watermark Embedding

Algorithm 1 shows the steps of the proposed watermark embedding algorithm. Step 2–22 performs the integer wavelet transform to embed the payload in the image, and creates the location map. Steps 24–36 calculates the pixel values of the watermarked image. The above can embed at most $(n-1)$ bits per row of the image matrix X, where each row has n columns. Hence, the embedding capacity of the algorithm is quite large and the visual quality is satisfactory because of high signal–to–noise ratio, as found in our experiment. The above algorithm can also be applied on an image already containing

an embedded watermark to embed further sets of watermarks. The data embedding capacity of the above algorithm can be improved further by directly embedding multiple bits of the payload in the difference values as guided by their *hiding ability*.

As a simple example, consider a 4×8 "image matrix" (with the pixel values in decimal):

$$X = \begin{bmatrix} 84 & 80 & 72 & 67 & 88 & 90 & 70 & 63 \\ 70 & 75 & 78 & 80 & 82 & 76 & 59 & 51 \\ 76 & 74 & 77 & 82 & 64 & 55 & 47 & 44 \\ 74 & 79 & 92 & 104 & 55 & 42 & 40 & 41 \end{bmatrix} \quad (8)$$

Suppose, we want to embed the payload bitstream $msg = 11011001110110100111$. When X and msg are input to the watermark embedding algorithm, the first loop (steps 5–22) runs $\log_2 8 = 3$ times. The transformation of the $avgdiff$ matrix (at step 21) over the iterations is as follows:

1st iteration:

$$avgdiff^{(1)} = \begin{bmatrix} 82 & 9 & 69 & 11 & 89 & -4 & 66 & 15 \\ 72 & -9 & 79 & -4 & 79 & 12 & 55 & 17 \\ 75 & 5 & 79 & -9 & 59 & 18 & 45 & 7 \\ 76 & -9 & 98 & -24 & 48 & 27 & 40 & -2 \end{bmatrix}$$

2nd iteration:

$$avgdiff^{(2)} = \begin{bmatrix} 75 & 9 & 26 & 11 & 77 & -4 & 47 & 15 \\ 75 & -9 & -13 & -4 & 67 & 12 & 49 & 17 \\ 77 & 5 & -4 & -9 & 52 & 18 & 14 & 7 \\ 87 & -9 & -22 & -24 & 44 & 27 & 8 & -2 \end{bmatrix}$$

3rd iteration:

$$avgdiff^{(3)} = \begin{bmatrix} 76 & 9 & 26 & 11 & -2 & -4 & 47 & 15 \\ 71 & -9 & -13 & -4 & 8 & 12 & 49 & 17 \\ 64 & 5 & -4 & -9 & 25 & 18 & 14 & 7 \\ 65 & -9 & -22 & -24 & 43 & 27 & 8 & -2 \end{bmatrix}$$

The final watermarked image matrix output by the algorithm is:

$$X_{wm} = \begin{bmatrix} 93 & 84 & 68 & 57 & 99 & 103 & 62 & 47 \\ 65 & 74 & 80 & 84 & 98 & 86 & 52 & 35 \\ 78 & 73 & 75 & 84 & 68 & 50 & 49 & 42 \\ 72 & 81 & 86 & 110 & 62 & 35 & 39 & 41 \end{bmatrix}$$

C. Watermark Extraction

The watermark extraction algorithm shown in Algorithm 2 works in exactly the reverse way compared to the embedding algorithm. Steps 3–24 extracts the embedded payload from the watermarked image, steps 26–39 retrieves the original image.

IV. Results

The above methodology was implemented in MATLAB and applied to a 256×256, 8 bits per pixel (bpp) grayscale version of the "Lena" image. Fig. 1 shows the original version of the image. Watermark payloads of different sizes varying from 0.1 bpp to 0.8 bpp were embedded in the image. Fig. 2 shows the watermarked image with a 32768–bit (0.5bpp) embedded payload. From these two images, it is evident that the proposed approach has minimal adverse effect on the visual quality of the image. To calculate the PSNR, first the *mean square error* (MSE) was calculated as:

Fig. 1. Original 256×256, 8bpp grayscale "Lena" image.

Fig. 2. Reversibly watermarked "Lena" with a 32768–bit (0.5bpp) embedded payload, and PSNR=35.93dB.

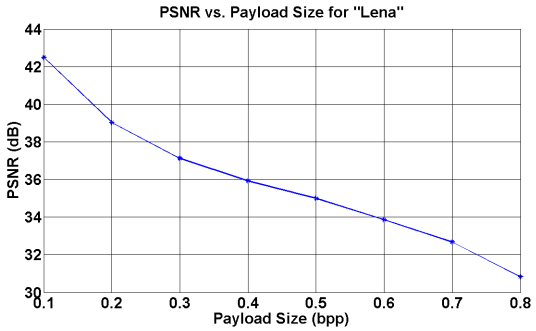

Fig. 3. Plot of PSNR vs. embedded payload size.

$$MSE = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{\left(X_{org}(i,j) - X_{wm}(i,j)\right)^2}{m \cdot n} \quad (9)$$

where $X_{org}(i,j)$ is the (i,j)–th pixel of the original image, and $X_{wm}(i,j)$ is the (i,j)–th pixel of the watermarked image, and m and n are the dimensions of the image (here each is

256). Then, PSNR is calculated as:

$$PSNR = 10\log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 10\log_{10}\left(\frac{255^2}{MSE}\right) \quad (10)$$

where MAX_I is the maximum possible pixel value of the image, which is 255 in this case because of the 8–bit grayscale nature of the image.

Fig. 3 shows a plot of the *peak signal–to–noise ratio* (PSNR, in dB) of the watermarked image against the embedded payload size (in bits per pixel). These results are comparable with state–of–the–art techniques such as [1], [6].

V. Conclusion

Reversible watermarking is an important class of techniques for digital content protection and authentication where it is possible to retrieve the original content with zero distortion. In this paper, we have proposed a high capacity reversible digital watermarking technique for images, where the spatial redundancy of images are utilized in embedding the watermark. The novelty of the proposed technique lies in the repeated application of the principle of *difference expansion* to decrease the number of average terms to a single average term and increase the number of difference terms, so that more bits of the payload can be embedded in the difference terms. This effectively increases the embedding capacity of the watermarked image. Experimental results on the common benchmark image "Lena" shows that the technique is capable of achieving good PSNR values even at large payload sizes.

References

[1] J. Tian, "Wavelet–based reversible watermarking for authentication", *Security and Watermarking of Multimedia Contents IV*, vol. 4675, pp. 679–690, 2002.

[2] Z. Ni, Y. Q. Shi, N. Ansari and W. Su, "Reversible data hiding", *Proceedings of the IEEE International Symposium of Circuits and Systems*, 2003.

[3] A. R. Calderbank, I. Daubechis, W. Sweldens and B. L. Yeo, "Wavelet Transforms that map integers to integers", *Applied and Computational Harmonic Analysis* vol. 5, pp. 332-369, 1998.

[4] R. C. Gonzalez and R. E. Woods, "Digital Image Processing (3rd edition)", *Pearson education*, 2009.

[5] R. C. Gonzalez, R. E. Woods and S. L. Eddins, "Digital Image Processing using MATLAB", *Pearson Education*, 2004.

[6] L. Luo, Z. Chen, M. Chen, X. Zeng and Z. Xiong, "Reversible image watermarking using interpolation technique", *IEEE Transactions on Information Forensics and Security* vol. 5, no. 1, pp. 187–193, Mar. 2010.