Signal Processing 94 (2014) 118–127

Contents lists available at SciVerse ScienceDirect

Signal Processing

journal homepage: www.elsevier.com/locate/sigpro

Reversibility improved data hiding in encrypted images ☆

Weiming Zhang *, Kede Ma, Nenghai Yu

School of Information Science and Technology, University of Science and Technology of China, Hefei 230027, China

ARTICLE INFO

Article history:
Received 16 November 2012
Received in revised form
20 June 2013
Accepted 22 June 2013
Available online 29 June 2013

Keywords:
Reversible data hiding
Image encryption
Privacy protection
Histogram shift

ABSTRACT

A novel reversible data hiding technique in encrypted images is presented in this paper. Instead of embedding data in encrypted images directly, some pixels are estimated before encryption so that additional data can be embedded in the estimating errors. A benchmark encryption algorithm (e.g. AES) is applied to the rest pixels of the image and a special encryption scheme is designed to encrypt the estimating errors. Without the encryption key, one cannot get access to the original image. However, provided with the data hiding key only, he can embed in or extract from the encrypted image additional data without knowledge about the original image. Moreover, the data extraction and image recovery are free of errors for all images. Experiments demonstrate the feasibility and efficiency of the proposed method, especially in aspect of embedding rate versus Peak Signal-to-Noise Ratio (PSNR).

1. Introduction

Reversible data hiding (RDH) has the capability to erase the distortion introduced by embedding step after cover restoration. It is an important property that can be applied to many scenarios, such as medical imagery, military imagery and law forensics. For this reason, RDH becomes a hot research topic and is extensively studied over the years.

Until now, many RDH techniques have been proposed based on three fundamental strategies: lossless compression-appending scheme [1,2], difference expansion (DE) [3,4] and histogram shift (HS) [5]. Some recent arts combined the three strategies to residuals of the image such as prediction errors (PE) [6–9] to achieve better performance. Almost all state-of-the-art RDH algorithms consist of two steps. The first step generates a host sequence with small entropy, i.e., the host has a sharp histogram which usually can be realized by using PE combined with the sorting technique [10] or pixel selection [11]. The second step reversibly embeds the message in the host sequence by modifying its histogram with methods like HS and DE. Optimal coding methods for modifying the histogram were proposed by Zhang et al. [12] and Lin and Chung [13]. On the other hand, some robust RDH methods have also been proposed [14–17].

Nowadays with the increasing demand of privacy protection, the ability to embed information in encrypted data will be useful in cloud computing [18]. In [19], the content owner encrypts the signs of host discrete cosine transform (DCT) coefficients. Different fingerprints are generated at the receiver side by decrypting only a subset of the coefficients with different decryption keys. In [20], during H.264/AVC compression, the intra-prediction mode, motion vector differences and DCT coefficients' signs are encrypted, while watermarking process proceeds on the DCT coefficients' amplitudes adaptively. In [21], a commutative watermarking and encryption system is presented based on a layered scheme and a key dependent transform domain. However, the data embedding is not reversible with the above-mentioned techniques [19–21]. As can be seen, there are some promising applications if RDH can be

☆ This work was supported in part by the Natural Science Foundation of China under Grants 61170234 and 60803155, by the Strategic and Piloted Project of CAS under Grant XDA06030601, and by the Funding of Science and Technology on Information Assurance Laboratory under Grant KJ-13-02.

* Corresponding author. Tel.: +86 551 3600683.
E-mail addresses: weimingzhang@yahoo.cn, zwmshu@gmail.com (W. Zhang), k29ma@uwaterloo.ca (K. Ma), ynh@ustc.edu.cn (N. Yu).

0165-1684/$ - see front matter Crown Copyright © 2013 Published by Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sigpro.2013.06.023

adopted in encrypted images. For instance, by such technique, we can embed notations into an encrypted medical image. With the notation, a server can manage the image or verify its integrity without knowing the image content, and thus the privacy of the patient is protected. On the other hand, a doctor, having the key, can decrypt and restore the image losslessly.

Since the entropy of encrypted images has been maximized and leaves no spare space for traditional RDH methods to exploit, the embedding step may not be possible by using standard RDH algorithms. Even so, some attempts have been made to accommodate additional data reversibly in encrypted images. In [22], Zhang divided the encrypted image into several blocks. By flipping 3 Least Significant Bits (LSBs) of a group of specific pixels, one bit message can be embedded into each block. Hong et al. [23] improved Zhang's method [22] by further exploiting the spatial correlation using a different estimation equation and side match technique. In order to extract data, the two methods rely on decrypted images which may be unknown for some cases. Aiming for separating data extraction from image decryption, Zhang [24] found the syndromes of a low-density parity check matrix to compress the LSBs of the encrypted image. By doing so, an extra space is created to append additional data. These techniques can only achieve low embedding capacity (achievable largest embedding rate) [22,23] or generate marked image with poor quality for high embedding capacity [24] and all of them are subject to some errors on data extraction and/or image restoration. Although the methods in [22,23] can eliminate errors by error-correcting codes, the pure embedding capacity will be further consumed. On the other hand, to locate and modify the LSBs of the encrypted image, these methods must depend on a special encryption scheme, that is, encrypting the bit planes of pixels by a stream cipher. However, most current popular image encryption techniques encrypt pixels as integers with block ciphers.

This paper proposes a novel RDH method in encrypted spatial images based on estimation technique. A large portion of pixels are utilized to estimate the rest before encryption, and then encrypted with a standard encryption algorithm. After that we encrypt the estimating errors with a special encryption scheme. By concatenating encrypted estimating errors and the large group of encrypted pixels, the ultimate version of encrypted image is formulated. The additional data can be embedded in the encrypted image by modifying the estimating errors. In general, the excellent performance can be achieved in three different prospects:

- The proposed method is completely reversible. That is, no error happens in data extraction and image recovery steps.
- The PSNR values of marked decrypted image are much higher than those previous methods [22–24] can achieve under given embedding rates.
- The extraction and decryption steps are independent, which are more natural and applicable.

The rest of this paper is organized as follows. The scheme of the proposed method is elaborated in Section 2. Abundant experimental results with the analysis of complete reversibility are presented in Section 3. We conclude our paper with a discussion in Section 4.

2. Proposed method

To enable data embedding in the encrypted image, all the three methods [22–24] try to space out room from the encrypted image directly, which follows the idea of compressing the encrypted image [25,26]. However, losslessly vacating room from the encrypted image is difficult and thus these techniques cannot achieve good image quality or realize complete reversibility. In this section, we propose a novel method to significantly improve the performance by reversing the order of encryption and vacating room. In the light of this idea, we empty out room prior to image encryption by shifting the histogram of estimating errors of some pixels and the emptied out room will be used for data hiding.

The proposed method is composed of four primary steps: vacating room and encrypting image, data hiding in the encrypted image, data extraction and image recovery. Two different schemes, extraction before decryption and decryption before extraction, are raised to cope with different applications. Fig. 1 illustrates the overview of the proposed method.

2.1. Vacating room and encrypting image

Without loss of generality, assume that the original $M \times N$ image $\mathbf{X}$ is an 8-bit gray-scale image, with pixel

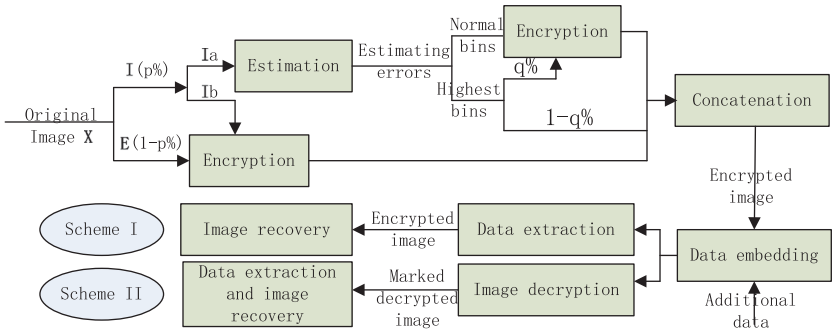

Fig. 1. The framework of proposed method.

$\mathbf{X}(i,j)\in[0,255]$, $1\le i\le M$, $1\le j\le N$. First, the content owner randomly selects $p\%$ pixels, denoted by $\mathbf{I}$, from $\mathbf{X}$ according to the encryption key and calculates their estimating values from the surrounding $1-p\%$ pixels, denoted by $\mathbf{E}$. Here, p is a secret integer determined by the content owner and usually smaller than 20.

The estimation of the pixel $\mathbf{X}(i,j)\in\mathbf{I}$ can be calculated through

$$\mathbf{X}'(i,j)=\left\lfloor\frac{\sum_{h=-1}^{1}\sum_{w=-1}^{1}\mathbf{X}(i+h,j+w)\mathrm{sign}(\mathbf{X}(i+h,j+w))}{\sum_{h=-1}^{1}\sum_{w=-1}^{1}\mathrm{sign}(\mathbf{X}(i+h,j+w))}\right\rfloor, \tag{1}$$

where sgn(∗) represents a sign function defined as

$$\mathrm{sgn}(\mathbf{X}(i,j))=\begin{cases}1 & \text{if } \mathbf{X}(i,j)\in\mathbf{E}\\ 0 & \text{if } \mathbf{X}(i,j)\in\mathbf{I}.\end{cases} \tag{2}$$

To circumvent the case that no pixels in $\mathbf{E}$ surround $\mathbf{X}(i,j)\in\mathbf{I}$ which makes Eq. (1) unappropriate and to ensure the accuracy of estimation, a threshold T ($1\le T\le 8$) is adopted. Only $\mathbf{X}(i,j)\in\mathbf{I}$ whose neighborhood includes at least T pixels in $\mathbf{E}$ is chosen to be estimated and denoted by $\mathbf{I}_a$. Correspondingly, pixel $\mathbf{X}(i,j)\in\mathbf{I}$ with less than T pixels in $\mathbf{E}$ surrounded, which is denoted by $\mathbf{I}_b$, is skipped and encrypted along with the $1-p\%$ pixels in $\mathbf{E}$. Fig. 2 illustrates the singular case associated with the normal case. It is true, in general, that estimation accuracy can benefit from the setting of large T. However, large T may limit the size of $\mathbf{I}_a$ and therefore decrease the embedding capacity. Through relevant experiments, we propose that $T=4$ can achieve a good balance between estimation accuracy and embedding capacity, which will be detailed in the next section.

Having obtained the estimating values, the content owner can calculate the estimating errors of pixels belonging to $\mathbf{I}_a$ through

$$e(i,j)=\mathbf{X}(i,j)-\mathbf{X}'(i,j). \tag{3}$$

The next step is to replace the gray value $\mathbf{X}(i,j)$ with its estimating error $e(i,j)$ for the pixels belonging to $\mathbf{I}_a$. A gray value $\mathbf{X}(i,j)$ ranging from 0 to 255 can be represented by 8 bits, $b_{i,j}(0), b_{i,j}(1),\ldots,b_{i,j}(7)$, such that

$$b_{i,j}(k)=\left\lfloor\frac{\mathbf{X}(i,j)}{2^k}\right\rfloor \bmod 2,\quad k=0,1,\ldots,7. \tag{4}$$

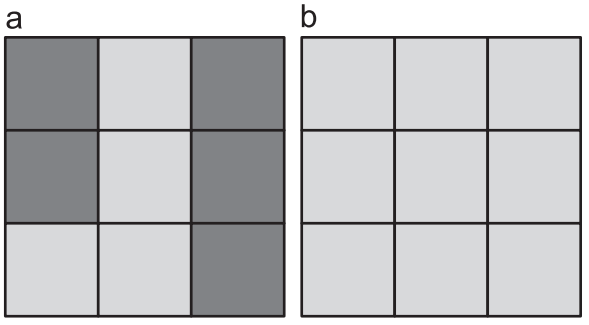

Fig. 2. The normal case (a) versus singular case (b) when calculating estimating values (dark gray area represents pixels in $\mathbf{E}$ and light gray area represent pixels belonging to $\mathbf{I}$).

However, the estimating error $e(i,j)$ can either be positive or negative. That is, only $e(i,j)$ falling into $[-127,127]$ can be correctly saved with 8 bits by using the most significant bit (MSB) $b_{i,j}(7)$ to stand for sign bit (for example, $b_{i,j}(7)=0$ indicates positive while $b_{i,j}(7)=1$ indicates negative) and the rest bits to represent the absolute value of $e(i,j)$. In order to enable reversible modifications of the histogram (which will be described later), we truncate the error values into $[-124,125]$. To do it, we modify the value of $e(i,j)$ ranging out of $[-124,125]$ as the difference between $e(i,j)$ and its corresponding boundary value. In other words, if $e(i,j)<-124$ we adjust it as $e(i,j)+124$; and if $e(i,j)>125$ we modify it as $e(i,j)-125$. And a location map O_1 is introduced to record the positions of such error values.

In the next step, the content owner creates embedding space by modifying the histogram of estimating errors in $\mathbf{I}_a$. First, find two highest bins in the histogram with their corresponding values denoted by P_1 and P_2, respectively. In general, the estimating errors satisfy a Laplacian distribution centered at zero so we select $P_1=-1$ and $P_2=0$ for the sake of simplicity. Second, search for two lowest bins and denote their corresponding values as Z_1 and Z_2, satisfying $Z_1<P_1$ and $P_2<Z_2$. With these 4 parameters, modification based on histogram shift can be operated via

$$e'(i,j)=\begin{cases}e(i,j)+\mathrm{sgn}'(e(i,j))\times 1 & \text{if } e(i,j)\in(Z_1,-2]\cup[1,Z_2)\\ e(i,j) & \text{otherwise,}\end{cases} \tag{5}$$

where $e'(i,j)$ represents shifted estimating error. sgn′(∗) is also a sign function specified as

$$\mathrm{sgn}'(e(i,j))=\begin{cases}-1 & \text{where } e(i,j)\in[Z_1,-2]\\ 1 & \text{where } e(i,j)\in[1,Z_2].\end{cases} \tag{6}$$

By shifting bins between Z_1+1 and -2 toward left and bins between 1 and Z_2-1 toward right, the bins -2 and 1 are emptied out. In practice, if the bin of Z_1 or Z_2 is not empty, the content owner should record the positions of error values equal to Z_1 or Z_2 in another location map O_2.

At this juncture it may be prudent to explain how to place location maps O_1 and O_2. For typical spatial images size of $M\times N$, we employ $\log_2\lceil pMN/100\rceil$ bits to stand for the length of the location map and the information of each position. Having decided their lengths, O_1 and O_2 are then embedded in pixels belonging to $\mathbf{E}$ by a traditional RDH method. For instance, we can select a portion of pixels in $\mathbf{E}$ to estimate the rest pixels and then get the estimating errors. The estimation here can be as simple as that used in Eq. (1). More specifically, only pixel in $\mathbf{E}$ whose neighborhood includes at least T pixels in $\mathbf{E}$ as well is chosen to be estimated by taking the average of surrounding pixels. After generating the estimating errors, difference expanding method in [4] or histogram shifting method in [7] can be applied to these errors to embed the location maps O_1 and O_2. Although this process may also cause overflow/underflow problems in $\mathbf{E}$, a self-contained RDH method can deal with such problems itself. For example, the positions of overflowing/underflowing elements are embedded in some reserved pixels by replacing their LSBs, and the original

LSBs of these pixels are reversibly embedded in the estimating errors.

After histogram shift, the content owner will provide shifted errors in $\mathbf{I}_a$ to the data hider for data embedding. This process would inevitably divulge the distribution of estimating errors in $\mathbf{I}_a$ to the data hider. To preserve such information from leakage, the content owner encrypts the estimating errors in the following manner.

First, with the encryption key the content owner randomly picks $q\%$ positions in $\mathbf{I}_a$, denoted by *Ind*. If $e'(i,j)\in[2,125]$ or if $e'(i,j)=0$ and $(i,j)\in Ind$, generate a pseudo-random number $r(i,j)\in[0,125]$ with the encryption key, and encrypt the error value by

$$E_1(e'(i,j)) = (r(i,j) + e'(i,j) \bmod 126) + 2. \tag{7}$$

By adding 2, we shift the encrypted value from [0,125] to [2,127], and thus 0 and 1 are vacated for representing the bits to be embedded. Note that the original error values equal to 126 and 127 have been truncated to 1 and 2 and their positions have been recorded in location map O_1, so these truncated values can be easily recovered after decryption.

If $e'(i,j)\in[-124,-3]$ or if $e'(i,j)=-1$ and $(i,j)\in Ind$, generate a pseudo-random number $r(i,j)\in[0,124]$ with the encryption key, and encrypt the error by

$$E_2(e'(i,j)) = -(r(i,j) - e'(i,j) \bmod 125) - 3. \tag{8}$$

In the above equation we shift the encrypted value from $[-124,0]$ to $[-127,-3]$ by subtracting -3, and thus -1 and -2 are vacated for representing the bits to be embedded. Note that the original error values equal to -125, -126 and -127 have been truncated to -1, -2 and -3 and their positions have been recorded in location map O_1, so these truncated values can also be easily recovered after decryption.

In summary, all the errors belonging to [2, 125] and $[-124,-3]$ and approximate $q\%$ errors equal to 0 and -1 are encrypted. Note that q is a secret integer held by the content owner, and thus all the frequencies of estimating errors, including 0 and -1, are protected.

One simple example on how to shift and encrypt the histogram of estimating errors is illustrated in Fig. 3. In this example, the error values are taken from $[-15,15]$, which can be represented by five bits. Before encryption, the errors have been truncated to $[-12,13]$, and we will replace the modular number 126 with 14 in (7) and replace the modular number 125 with 13 in (8). As shown in Fig. 3(a), two lowest bins in the original histogram are denoted by -8 and 8, both of which have zero frequencies. The values belonging to [1, 7] are shifted toward the right and the values belonging to $[-2,-7]$ are shifted toward the left, which results in the shifted histogram (Fig. 3(b)). Next, we select a secret integer, e.g. $q=20$, and generate a random index *Ind* that includes 20% indexes of the errors. Finally, we encrypt about 20% error values equal to 0 and -1 according to the index *Ind*, and encrypt all the error values belonging to $[-3,-15]$ and [2, 15]. By comparing Fig. 3(a) with Fig. 3(c), it can be seen that all the frequencies in Fig. 3(a) are concealed after encryption.

After shifting and encrypting the histogram, the rest bin 0 and bin -1 can be used to embed information. We can embed the message bit b into the error value $e'(i,j)$ equal to 0 or -1 by the following equation:

$$e''(i,j) = \begin{cases} e'(i,j)+b & \text{if } e'(i,j)=0 \\ e'(i,j)-b & \text{if } e'(i,j)=-1. \end{cases} \tag{9}$$

The bit b can be extracted from $e''(i,j)$ by

$$b = \begin{cases} 0 & \text{if } e''(i,j)=0 \text{ or } -1 \\ 1 & \text{if } e''(i,j)=1 \text{ or } -2. \end{cases} \tag{10}$$

The error value $e'(i,j)$ can be recovered by

$$e'(i,j) = \begin{cases} e''(i,j) & \text{if } e''(i,j)=0 \text{ or } -1 \\ e''(i,j)-1 & \text{if } e''(i,j)=1 \\ e''(i,j)+1 & \text{if } e''(i,j)=-2. \end{cases} \tag{11}$$

We reserve first several 0 and -1 to embed the length of $\mathbf{I}_a$. For a 512×512 image, sixteen 0 and -1 are enough to accommodate the length information.

The next step is to rearrange encrypted errors in $\mathbf{I}_a$ to the top of the image followed by the pixels in $\mathbf{I}_b$ and $\mathbf{E}$ as shown in Fig. 4, which protects the position information of image from being divulged. Finally, the content owner encrypts the pixels in $\mathbf{I}_b$ and $\mathbf{E}$ using a secure encryption algorithm such as AES with the encryption key. Fig. 4 shows the state of final encrypted image, denoted by $\mathbf{X}_E$. Without the encryption key, a data hider or a third party is difficult to acquire details of the original image.

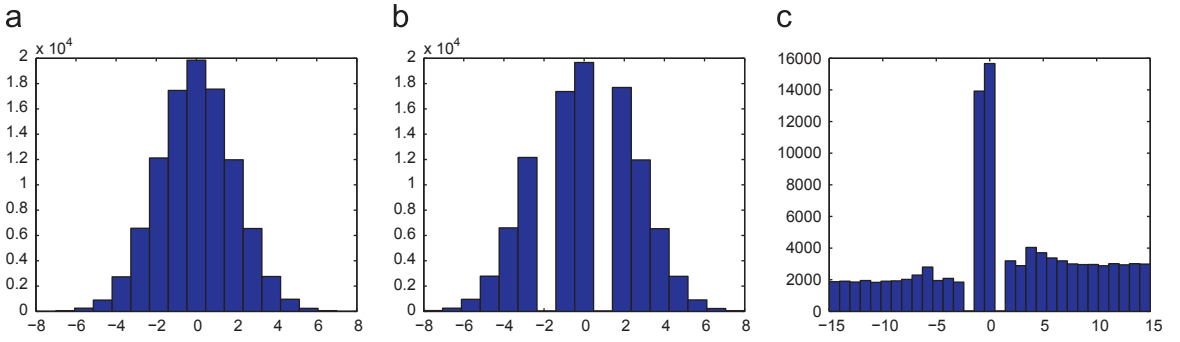

Fig. 3. (a) Original histogram, (b) shifted histogram and (c) encrypted histogram

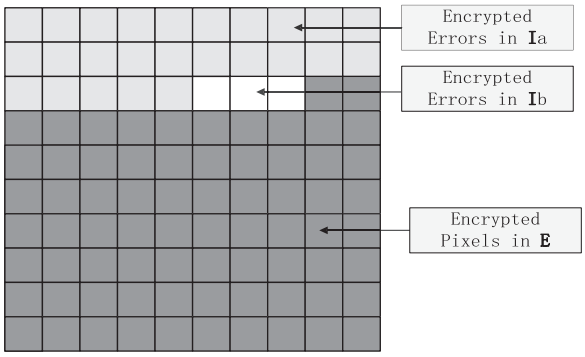

Fig. 4. The final version of the encrypted image $\mathbf{X}_E$.

2.2. Data hiding in the encrypted image

When a data hider, perhaps a server, receives the encrypted image, he can embed some information into it for the purpose of media notation or integrity authentication if he possesses the data hiding key.

First, the data hider reads the length of $\mathbf{I}_a$ by extracting the first sixteen $0, 1, -1$ and -2 from the top of the image and decoding them with Eq. (10). Second, he encrypts the data to be embedded with the data hiding key. Third, he embeds the encrypted information into the rest error values in $\mathbf{I}_a$ that are equal to 0 and -1 with Eq. (9). Finally, with the data hiding key, the data hider randomly permutes the positions of elements of $\mathbf{X}_E$, and thus a potential attacker cannot extract the embedded data.

2.3. Data extraction and image recovery

2.3.1. Scheme I: data extraction before image decryption

Consider the scenario that a low level server, who is not granted the authority to get access to original images in consideration of protecting clients' privacy, may have the duty to note and modify the personal information in corresponding encrypted images as well as to verify images' integrity. A possible solution to this case must guarantee the independence between data extraction and image decryption.

In the proposed method, when the server acquires the data hiding key, he first permutes back the elements of $\mathbf{X}_E$ and gets $\mathbf{I}_a$ that has been put into the top of the image. Thus, he can extract the encrypted bits from $\mathbf{I}_a$ with Eq. (10), and then decrypt the extracted binary sequence to get the embedded information. By Eq. (11), he can further restore the encrypted image provided by the content owner and embed new information all over again.

Scheme I is especially suitable for medical and military image management, where images are confidential and need to be encrypted. Decryption and restoration of the image in Scheme I is similar to that in Scheme II, which will be described in the next subsection.

2.3.2. Scheme II: image decryption before data extraction

In Scheme I both data embedding and extraction are manipulated in encrypted domain. On the other hand, there exists another situation that the user wishes to decrypt the image first and extract the data from the decrypted image when it is needed. So we propose Scheme II, image decryption before data extraction. The following is an application scenario for this scheme. Assume Alice outsourced her images to a cloud server, and the images are encrypted to protect their contents. Into the encrypted images, the cloud server embedded some notation, including the identities of the images' owner and the cloud server and time stamps, to manage the encrypted images, as proposed in [18]. Note that the cloud server has no right to do any permanent damage to the images. Now an authorized user, Bob, downloaded and decrypted the images. Bob hoped that the decrypted images still include the notation, which can be used to trace the source of the data. In this case, Scheme II is useful. The algorithm of scheme II is described as follows.

Generating the marked decrypted image: To formulate the marked decrypted image $\mathbf{X}''$, the content owner should do the following five steps:

- *Step*1: With the shared data hiding key, permute back the order of the elements in X_E. Read the length of $\mathbf{I}_a$ from the first sixteen values of X_E equal to $0, 1, -1$, and -2, and then sequentially decrypt all the error values in $\mathbf{I}_a$ that belongs to $[2, 127]$ and $[-127, -3]$. For simplicity, the encrypted error is uniformly denoted by $E(e'(i,j))$. If an encrypted error $E(e'(i,j))\in[2, 127]$, decrypt it as
$$e'(i,j) = E(e'(i,j)) - 2 - r(i,j) \bmod 126. \tag{12}$$
If an encrypted error $E(e'(i,j))\in[-127, -3]$, decrypt it as
$$e'(i,j) = -(-E(e'(i,j)) - 3 - r(i,j) \bmod 125). \tag{13}$$
In (12) $r(i,j)$ is a pseudo-random number belonging to $[0, 125]$; in (13) $r(i,j)$ is a pseudo-random number belonging to $[0, 124]$. The pseudo-random number is generated with the encryption key. The pixels in $\mathbf{I}_b$ and $\mathbf{E}$ are also decrypted with the encryption key.
- *Step*2: Generate $p\%$ random positions of an $M \times N$ image with the encryption key, and sequentially put the pixels of $\mathbf{E}$ into the other $1-p\%$ positions. With the number of surrounding pixels in $\mathbf{E}$, the positions of pixels in $\mathbf{I}_a$ and $\mathbf{I}_b$ can easily be identified and put back.
- *Step*3: Extract the location map O_1 from $\mathbf{E}$. For a location $(i,j)\in O_1$, if $e'(i,j) > 0$, it is adjusted as $e'(i,j) = e'(i,j) + 125$ and if $e'(i,j) < 0$, it is adjusted as $e'(i,j) = e'(i,j) - 124$.
- *Step*4: Calculate estimating values $\mathbf{X}'(i,j)$ of $\mathbf{X}(i,j)\in\mathbf{I}_a$ through (1) and obtain the marked pixel $\mathbf{X}''(i,j)$ such that
$$\mathbf{X}''(i,j) = \mathbf{X}'(i,j) + e'(i,j). \tag{14}$$
Note that overflow/underflow maybe occurs, because $e'(i,j)$ is the modified error value. We should record the positions (i,j) such that $\mathbf{X}''(i,j) = 256$ or -1 in the third location map O_3, and adjust the pixel value from 256 to 255 or from -1 to 0. The location map O_3 is also embedded into $\mathbf{E}$ with a traditional RDH method.
- *Step*5: Substitute $e'(i,j)$ with $\mathbf{X}''(i,j)$.

As previously stated, the difference between the original $\mathbf{X}$ and watermarked $\mathbf{X}''$ is imperceptible, since the whole

process only modifies the gray values of $\mathbf{X}(i,j)\in\mathbf{I}_a$ at most 1. When the threshold T is relatively small, the PSNR values of the marked image $\mathbf{X}''$ versus the original image $\mathbf{X}$ can be predicted by

$$\mathrm{PSNR}=10\times\log_{10}\left(\frac{255\times255}{p\%\times(1-q\%)}\right). \tag{15}$$

Data extraction and image restoration: After generating the marked decrypted image, the content owner can further extract the data and recover the original image. The process is essentially similar to that of traditional RDH methods. The following outlines the specific steps:

- *Step*1: Locate positions of pixels in $\mathbf{E}$ according to the encryption key, and implement RDH algorithm reversely to extract location maps, O_1, O_2 and O_3, and restore the original pixels belonging to $\mathbf{E}$.
- *Step*2: Locate positions of pixels in $\mathbf{I}_a$ and calculate their estimating values $\mathbf{X}'(i,j)$ through (1). By reversing the function (14), we get error values

$$e'(i,j)=\mathbf{X}''(i,j)-\mathbf{X}'(i,j), \tag{16}$$

  which should be further adjusted according to O_1 and O_3. For a position $(i,j)\in O_1$, if $e'(i,j)>0$, it is adjusted as $e'(i,j)=e'(i,j)-125$ and if $e'(i,j)<0$, it is adjusted as $e'(i,j)=e'(i,j)+124$. For a position $(i,j)\in O_3$, if $\mathbf{X}''(i,j)=0$, it is adjusted as $e'(i,j)=e'(i,j)-1$ and if $\mathbf{X}''(i,j)=255$, it is adjusted as $e'(i,j)=e'(i,j)+1$.
- *Step*3: Generate the index *Ind* with the encryption key, which includes $q\%$ random positions of $\mathbf{I}_a$. Extract data by decoding the error values 0 and -1 with Eq. (10) but skip all the error values whose positions belong to *Ind*.
- *Step*4: Recover the estimating errors via (17).

$$e(i,j)=\begin{cases} e'(i,j)-\mathrm{sgn}'(e'(i,j))\times 1 & \text{if } e'(i,j)\in[Z_1,-2]\cup[1,Z_2] \\ e'(i,j) & \text{otherwise}\end{cases} \tag{17}$$

  Furthermore, adjust the error values according to O_1 and O_2. For a position $(i,j)\in O_1$, if $e(i,j)\geq 0$, it is adjusted as $e(i,j)=e(i,j)+125$ and if $e(i,j)<0$, it is adjusted as $e(i,j)=e(i,j)-124$. For a position $(i,j)\in O_2$, if $e(i,j)=Z_1+1$, it is adjusted as $e(i,j)=Z_1$ and if $e(i,j)=Z_2-1$, it is adjusted as $e(i,j)=Z_2$.
- *Step*5: Recover the original gray values of $\mathbf{X}(i,j)$ whose corresponding $e(i,j)\in\mathbf{I}_a$ by working out

$$\mathbf{X}(i,j)=\mathbf{X}'(i,j)+e(i,j). \tag{18}$$

- *Step*6: Replace watermarked gray value $\mathbf{X}''(i,j)$ with its original value $\mathbf{X}(i,j)$, and the original image $\mathbf{X}$ is successfully recovered.

3. Experimental results

Standard test image, Lena ($512\times512\times8$), shown in Fig. 5(a), is adopted to demonstrate the feasibility of the proposed method. The complete reversibility (CR) of the proposed algorithm has been verified due to the uniformity between the restored image and the original image. The encrypted image is displayed in Fig. 5(b), and Fig. 5(c) is the decrypted image containing watermark by setting $T=4$, $p=10$ and $q=10$. The recovery version is illustrated in Fig. 5(d). We define embedding rate (ER) as bpp (bits per pixel), that is, the average number of message bits carried by each pixel. In the example shown in Fig. 5, the embedding rate is 0.02 bpp.

Before comparing the proposed method with state-of-the-art algorithms [22–24], it is worth discussing three parameters p, q and T that affect the performance of the proposed approach. Both parameters p and q are determinant of embedding capacities. In other words, larger p or q corresponds to higher embedding capacity. Thus in practice, we adjust p to adapt to different applications. And for simplicity, we restrict $1\leq p\leq 20$. Given a certain embedding rate, the user can randomly select a p ($p\in[1,20]$) as long as the corresponding embedding capacity is larger than the required embedding rate. After that, the user can randomly select a q to ensure large enough capacity and conceal the frequencies of estimating errors equal to 0 and -1. For instance, assume that after determining p the total number of 0 and -1 is 1016 and we want to embed 100 bits of messages. Thus, we can randomly select q in the range $[0,90)$. Note that 16 bits have been used for the length of $\mathbf{I}_a$.

The third parameter T is concerned with estimation accuracy and embedding capacity which are contradictory to each other as stated above. Meanwhile, the estimation accuracy in a way determines the lengths of location maps O_1, O_2 and O_3. In this paper, we investigate situations that $T\geq4$ under different p, and determine its value experimentally. In order to consider the statistical results on different images, a popular gray-scale image database [27] from which we randomly select 50 images is used to test our algorithm in terms of parameter T. Table 1 tabulates the

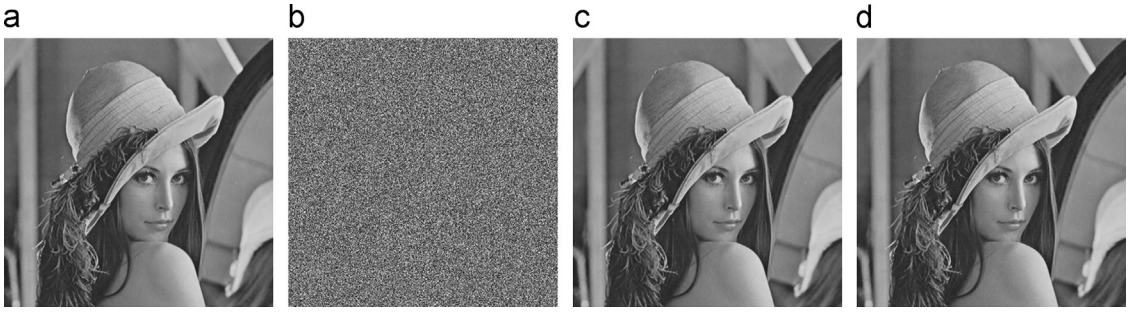

Fig. 5. (a) Original image Lena, (b) encrypted image with embedding rate 0.02 bpp, (c) marked decrypted image with PSNR=58.70 dB and (d) recovery version.

Table 1
The average lengths of three location maps and embedding rate (ER) under different T and p ($q=10$) on 50 images of database [27].

p(%)		1(%)	5(%)	10(%)	15(%)	20(%)
$T=4$	ER (bpp)	0.0019	0.0097	0.020	0.029	0.039
	O_1 (bits)	4.50	24.48	54.96	95.76	138.36
	O_2 (bits)	0.66	1.02	4.38	6.90	9.36
	O_3 (bits)	73.08	336.24	679.26	980.94	1276.50
$T=5$	ER (bpp)	0.0019	0.0097	0.020	0.029	0.037
	O_1 (bits)	4.50	24.48	54.96	90.72	124.80
	O_2 (bits)	0.66	1.02	4.02	6.54	8.64
	O_3 (bits)	73.08	335.16	668.10	958.26	1216.40
$T=6$	ER (bpp)	0.0019	0.0097	0.019	0.026	0.031
	O_1 (bits)	4.50	24.48	52.80	79.92	95.40
	O_2 (bits)	0.66	1.02	4.02	5.52	5.88
	O_3 (bits)	73.44	335.88	640.74	875.82	1020.20
$T=7$	ER (bpp)	0.0019	0.0092	0.016	0.019	0.020
	O_1 (bits)	4.50	23.04	41.34	52.68	56.70
	O_2 (bits)	0.66	1.02	3.36	3.06	2.04
	O_3 (bits)	74.88	316.80	539.28	644.70	649.80
$T=8$	ER (bpp)	0.0017	0.0064	0.0083	0.0079	0.0067
	O_1 (bits)	4.14	15.12	19.44	19.38	17.28
	O_2 (bits)	0.66	0.66	1.02	1.02	0.66
	O_3 (bits)	67.02	219.60	275.16	249.24	219.36

Table 2
The average performance of the proposed method on 50 images of database [27].

p(%)	1(%)	5(%)	10(%)	15(%)	20(%)
ER (bpp)	0.0019	0.0097	0.020	0.029	0.039
PSNR (dB)	68.43	61.35	58.31	56.54	55.34

results of such statistical analysis. As expected, the average lengths of three location maps and average embedding capacities decrease with increasing T. An important point here is that three location maps are embedded in $\mathbf{E}$ which is independent from the additional data embedded in $\mathbf{I}_a$. In other words, the embedding capacity of the proposed method is not reduced with the increasing sizes of three location maps. On the other hand, since the largest length of location map is 1276.50 bits ($T=4$ and $p=20$) which is much smaller compared with the size of $\mathbf{E}$, to keep a large embedding capacities, we fix $T=4$ in practical applications.

With the setting $p\in\{1,5,10,15,20\}$, $q=10$ and $T=4$, the average performance of the proposed method is summarized in Table 2. It can be observed that the proposed method achieves a wide range of embedding rate for acceptable PSNRs. In addition, the experimental results here support the theoretic PSNR values calculated by (15).

Now, it is necessary to compare the proposed method with previous works [22–24]. First, to clearly understand the concept CR, we quantify it by introducing average error pixels A_{ep}, which is defined as

$$A_{ep}=\frac{U}{Y}, \tag{19}$$

where Y is the number of images for testing and U is the total number of pixels that cannot be correctly recovered in Y images. As previous, we randomly select 50 natural images in [27] and present comparison results of proposed method with Zhang's [24] in Table 3. It is clear that the A_{ep} of proposed method equal to zero verifies its CR property. However, the method in [24] cannot guarantee CR at any embedding rate with $A_{ep}>10\,000$. Further, the proposed method can keep PSNR above 55 dB when embedding rate reaches 0.04 bpp. On the contrary, the algorithm in [24] suffers from poor visual quality for large embedding rate with PSNR values around 38 dB.

One reason previous methods [22,23] are not included in the above comparison is that they cannot separate data extraction from image decryption that limits their applications in practice. Another reason is that, for the methods in [22,23], the error occurs in data extraction and image recovery can be erased with the help of error-correcting codes. However, the actual embedding capacity should be reduced to

$$Cap_a=Cap_r(1-H(\rho)), \tag{20}$$

where Cap_r indicates the raw embedding capacity and $H(\rho)$ is the binary entropy function with error rate ρ. Take Zhang's method [22] as instance. The raw embedding capacity it can achieve for image size 512×512 is 0.016 bpp associated with minimum block size 8×8. With a typical error rate 7.00%, the corresponding actual embedding capacity is reduced to 0.010 bpp.

We select 4 standard images from [27] which include “Lena”, “Airplane”,“Barbara” and “Baboon” and plot the PSNR results under given embedding rates. Out of fairness, we modify the methods in [22,23] with error-correcting codes to eliminate errors. As for the algorithm in [24], we only choose those results with a significantly high probability of successful data extraction and perfect image recovery to draw the curves. From Fig. 6, we can notice

Table 3
Average performance comparison of the proposed method and Zhang's [24] on 50 images of database [27].

ER (bpp)		0.005	0.01	0.02	0.03	0.04
PSNR (dB)	Proposed	61.62	61.35	58.31	56.54	55.34
	Zhang's [24]	45.22	41.04	38.94	38.49	38.05
A_{ep} (bits)	Proposed	0	0	0	0	0
	Zhang's [24]	29 838	10 733	20 837	34 622	46 563

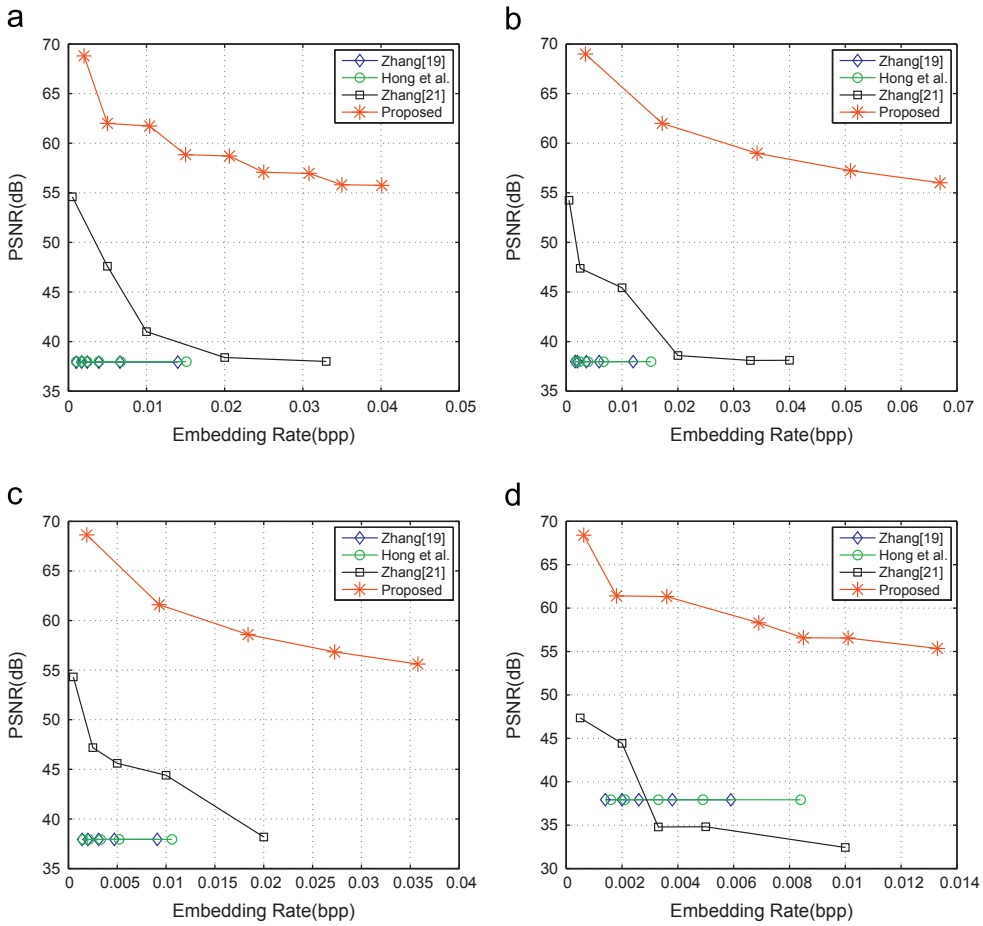

Fig. 6. Performance comparison of different methods in terms of PSNR under given embedding rate. (a) Lena, (b) Airplane, (c) Barbara and (d) Baboon.

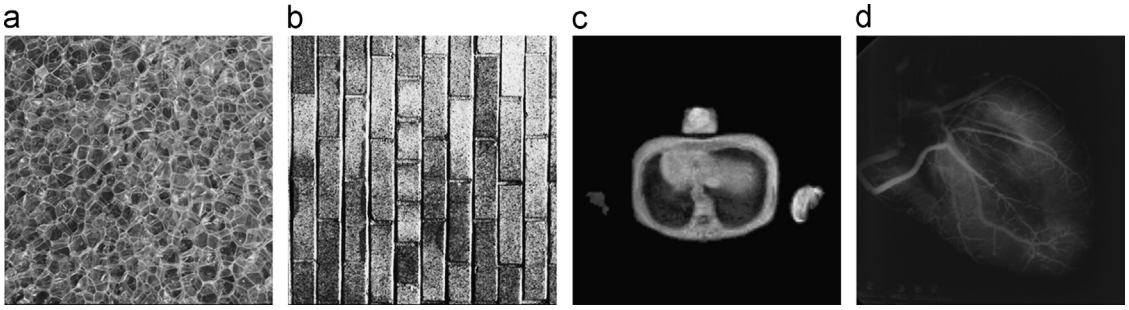

Fig. 7. Test images. (a) Texture1113, (b) Texture1212, (c) Mr2 and (d) Heart.

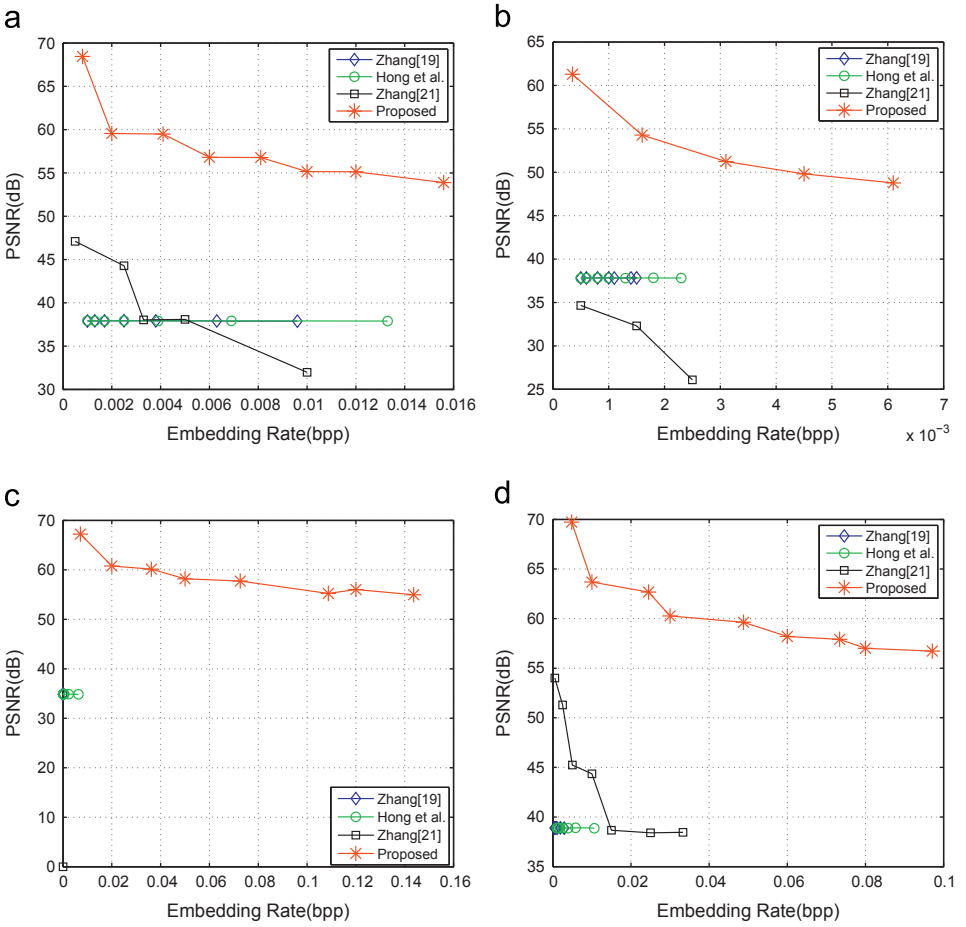

Fig. 8. Performance comparison under textural images and medical images. (a) Texture1113, (b) Texture1212, (c) Mr2 and (d) Heart.

Table 4
Average computational times (s) on 50 images of database [27].

p(%)	1(%)	5(%)	10(%)	15(%)	20(%)
Encryption (s)	0.41	0.73	1.12	1.50	1.90
Data hiding (s)	0.048	0.12	0.22	0.31	0.41
Decryption (s)	0.39	0.67	1.03	1.38	1.74

that the top curve represents the proposed method and its PSNR values are 15–18 dB higher than the existing reversible techniques [22–24].

Furthermore, we select two typical textural images from [27] and two typical medical images from [28] as test images that are shown in Fig. 7. The curve of PSNRs versus embedding rates is plotted in Fig. 8 which shows that, for the test image "Mr2", Zhang's method [24] cannot achieve an available embedding rate, and for all the test images, the proposed method reaches significantly higher PSNRs with larger achievable embedding rates than the other three methods.

In addition, another experiment has been conducted to measure the computational complexity of the proposed method and how it allocates the complexity to three separated steps: generation of encrypted image, data hiding process and generation of marked decrypted image. The results are reported in Table 4, where the speed is measured in seconds based on Matlab2012b implementations on a computer with Intel Core2 Quad CPU Q9400 processor at 2.67 GHz. As can be observed generally, the complexity of the proposed method increases with the embedding rates that are determined by p. And the proposed method allocates high complexity on encryption and decryption steps relative to data hiding process. In fact, the percentage of time spent on the data hiding process ranges from 6% to 10% of the overall method. Therefore, the proposed method is ideal for encrypted images management, where the primary manipulation is data hiding and extraction.

4. Conclusion and discussion

This paper proposes a RDH method for encrypted images by shifting the encrypted histogram of predicted errors, and achieves excellent performance in three aspects: complete reversibility, higher PSNR under given embedding rate, separability between data extraction and image decryption. Our method can work on two schemes independently in order to suit different application

prospects by extracting the data from the encrypted image or from the decrypted image.

References

[1] J. Fridrich, M. Goljan, Lossless data embedding for all image formats, in: SPIE Proceedings of Photonics West, Electronic Imaging, Security and Watermarking of Multimedia Contents, vol. 4675, San Jose, 2002, pp. 572–583.

[2] M. Celik, G. Sharma, A. Tekalp, E. Saber, Lossless generalized-LSB data embedding, IEEE Transactions on Image Processing 14 (2) (2005) 253–266.

[3] J. Tian, Reversible data embedding using a difference expansion, IEEE Transactions on Circuits System and Video Technology 13 (8) (2003) 890–896.

[4] D.M. Thodi, J.J. Rodriguez, Expansion embedding techniques for reversible watermarking, IEEE Transactions on Image Processing 16 (3) (2007) 721–730.

[5] Z. Ni, Y. Shi, N. Ansari, S. Wei, Reversible data hiding, IEEE Transactions on Circuits and Systems for Video Technology 16 (3) (2006) 354–362.

[6] P. Tsai, Y.C. Hu, H.L. Yeh, Reversible image hiding scheme using predictive coding and histogram shifting, Signal Processing 89 (2009) 1129–1143.

[7] L. Luo, Z. Chen, M. Chen, et al., Reversible image watermarking using interpolation technique, IEEE Transactions on Information Forensics and Security 5 (March (1)) (2010) 187–193.

[8] Y. Hu, H. Lee, J. Li, DE-based reversible data hiding with improved overflow location map, IEEE Transactions on Circuits and Systems for Video Technology 19 (February (2)) (2009) 250–260.

[9] W. Hong, T-S. Chen, C-W. Shiu, Reversible data hiding for high quality images using modification of prediction errors, Journal of Systems and Software 82 (2009) 1833–1842.

[10] V. Sachnev, H.J. Kim, J. Nam, S. Suresh, Y. Shi, Reversible watermarking algorithm using sorting and prediction, IEEE Transactions on Circuits and Systems for Video Technology 19 (July (7)) (2009) 989–999.

[11] X. Li, B. Yang, T. Zeng, Efficient reversible watermarking based on adaptive prediction-error expansion and pixel selection, IEEE Transactions on Image Processing 20 (December (12)) (2011) 3524–3533.

[12] W. Zhang, B. Chen, N. Yu, Improving various reversible data hiding schemes via optimal codes for binary covers, IEEE Transactions on Image Processing 21 (6) (2012) 2991–3003.

[13] S.-J. Lin, W-H. Chung, The scalar scheme for reversible information-embedding in gray-scale signals: capacity evaluation and code constructions, IEEE Transactions on Information Forensics and Security 7 (4) (2012) 1155–1167.

[14] X. Gao, L. An, Y. Yuan, D-C. Tao, X-L. Li, Lossless data embedding using generalized statistical quantity histogram, IEEE Transactions on Circuits and Systems for Video Technology 21 (August (8)) (2011) 1061–1070.

[15] L. An, X. Gao, X-L. Li, D-C. Tao, C. Deng, J. Li, Robust reversible watermarking via clustering and enhanced pixel-wise masking, IEEE Transactions on Image Processing 21 (August (8)) (2012) 3598–3611.

[16] X. Gao, L. An, X. Li, D. Tao, Reversibility improved lossless data hiding, Signal Processing 89 (October (10)) (2009) 2053–2065.

[17] L. An, X. Gao, Y. Yuan, D. Tao, Robust lossless data hiding using clustering and statistical quantity histogram, Neurocomputing 77 (February (1)) (2012) 1–11.

[18] K. Hwang, D. Li, Trusted cloud computing with secure resources and data coloring, IEEE Internet Computing 14 (September–October (5)) (2010) 14–22.

[19] D. Kundur, K. Karthik, Video fingerprinting and encryption principles for digital rights management, Proceedings of the IEEE 92 (2004) 918–932.

[20] S. Lian, Z. Liu, Z. Ren, H. Wang, Commutative encryption and watermarking in video compression, IEEE Transactions on Circuits and Systems for Video Technology 17 (6) (2007) 774–778.

[21] M. Cancellaro, F. Battisti, M. Carli, G. Boato, F.G.B. Natale, A. Neri, A commutative digital image watermarking and encryption method in the tree structured Haar transform domain, Signal Processing: Image Communication 26 (1) (2011) 1–12.

[22] X. Zhang, Reversible data hiding in encrypted images, IEEE Signal Processing Letters 18 (4) (2011) 255–258.

[23] W. Hong, T. Chen, H. Wu, An improved reversible data hiding in encrypted images using side match, IEEE Signal Processing Letters 19 (4) (2012) 199–202.

[24] X. Zhang, Separable reversible data hiding in encrypted image, IEEE Transactions on Information Forensics and Security 7 (2) (2012) 826–832.

[25] M. Johnson, P. Ishwar, V.M. Prabhakaran, D. Schonberg, K. Ramchandran, On compressing encrypted data, IEEE Transactions on Image Processing 52 (10) (2004) 2992–3006.

[26] W. Liu, W. Zeng, L. Dong, Q. Yao, Efficient compression of encrypted grayscale images, IEEE Transactions on Image Processing 19 (4) (2010) 1097–1102.

[27] Miscellaneous gray level images, available: ⟨http://decsai.ugr.es/cvg/dbimagenes/g512.php⟩.

[28] The USC-SIPI Image Database, available: ⟨http://sipi.usc.edu/database/database.php?volume=textures⟩.