Signal Processing 133 (2017) 40–51

Contents lists available at ScienceDirect

Signal Processing

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

Binary-block embedding for reversible data hiding in encrypted images

Shuang Yi, Yicong Zhou*

Department of Computer and Information Science, University of Macau, Macau 999078, China

ARTICLE INFO

Keywords:
Binary-block embedding (BBE)
Reversible data hiding
Image encryption
Encrypted domain

ABSTRACT

This paper first introduces a binary-block embedding (BBE) method to embed secret data in a binary image. Using BBE, we propose an algorithm for reversible data hiding in encrypted images (BBE-RDHEI). It uses BBE to embed binary bits in lower bit-planes of the original image into its higher bit-planes such that the lower bit-planes can be reserved for hiding secret data in subsequent processes. BBE-RDHEI employs a bit-level scrambling process after secret data embedding to spread embedded secret data to the entire marked encrypted image so that it can prevent secret data from loss. A security key design mechanism is proposed to enhance the security level of BBE-RDHEI. The processes of BBE-RDHEI are fully reversible. The secret data and original image can be reconstructed independently and separately. Experiments and comparisons show that BBE-RDHEI has an embedding rate nearly twice larger than the state-of-the-art algorithms, generates the marked decrypted images with high quality, and is able to withstand the brute-force, differential, noise and data loss attacks.

1. Introduction

Reversible data hiding (RDH) is a technique that slightly alters digital media (e.g. images or videos) to embed secret data while the original digital media can be completely recovered without any error after the hidden messages have been extracted [1]. It is quite useful for various applications in military, medical science or law enforcement, where the original images or videos should not be damaged. A number of RDH methods were proposed in recent years. Histogram shifting (HS) shifts several or the maximum points in histogram bins of the original image to reserve spare space for data embedding [1]. To improve the embedding capacity, prediction-error based HS algorithms were introduced [2–4]. Difference expansion (DE) [5–7] as another type of RDH divides the image into pixel pairs and embeds secret data into the expanded difference values. Integer transforms have been used to modify the values of pixel pairs to embed secret data [8–10]. These RDH methods need the redundancy information of image pixels in original images to embed secret data, such as the statistic or difference information of pixel pairs. They are not suitable for encrypted images that are noise-like and have no redundancy information available.

Recently, reversible data hiding in encrypted images (RDHEI) has attracted people's attention. It aims to protect both the original images and secret data simultaneously. For example, the content owner intends to store an original image in the Cloud that is hosted by a third party. To prevent the content of the original image from being exposed to the third party, the content owner encrypts the image before sending it to the Cloud. Meanwhile, the system administrator of the third party is able to add some notations to the encrypted image without knowing its original content. Depending on whether the data extraction and image recovery processes can be performed separately, existing RDHEI methods can be classified into joint and separate methods.

For joint methods, Peuch et al. [11] first encrypted each block of the original image by the advanced encryption standard (AES) and then embedded one bit of the secret data into each encrypted block by bit substitution. The encrypted image embedded with secret data is called the marked encrypted image. Secret data extraction is just to obtain the bits in the substituted positions. Original image recovering is accomplished by analyzing the local standard deviation of the marked encrypted image during the decryption procedures. This algorithm has a limited payload and yields the decrypted image with low quality. Another joint RDHEI algorithm proposed by Zhang [12] encrypts the original image using bit-level XOR, and then embeds one bit of secret data into each block of the encrypted image by shifting the three least significant bits (LSBs) of half pixels within the block. This algorithm may suffer from incorrect results of data extraction and image recovering in the non-smoothness regions in the image when the block size is relatively small (e.g., 8×8). Hong et al. [13] proposed an improved version of this algorithm by modifying its smoothness measurement function. The error rate of data extraction is reduced for small block sizes. In Wu et al.'s joint method [14], one bit of the secret data is embedded by flipping the i^{th} ($1 \le i \le 6$) bit of pixels in a

* Corresponding author.
E-mail address: yicongzhou@umac.mo (Y. Zhou).

http://dx.doi.org/10.1016/j.sigpro.2016.10.017
Received 13 May 2016; Received in revised form 6 September 2016; Accepted 12 October 2016
Available online 25 October 2016

certain group. This method also may suffer from incorrect results of data extraction and image recovery.

To allow the receiver with different privileges to obtain different contents (the secret data, the original image or both) from the marked encrypted image, researchers devote themselves to develop separable RDHEI methods. Zhang et al. [15,16] proposed two separable methods that compress the encrypted image to accommodate secret data. In Wu et al.'s separable method [14], one bit of the secret data is embedded by replacing the i^{th} ($i \geq 7$ for the later one) bit of pixels in a certain group. Secret data extraction and image recovering are using the prediction error. Compared with algorithms in [12,13,15], the methods in [14] reduce the number of incorrectly extracted secret data bits and improve the visual quality of the marked decrypted image. Qian [17] proposed a separable RDHEI algorithm using n-nary histogram modification. However, it results in Salt & Pepper noise in the marked encrypted images. Besides, instead of working on the spatial domain, Qian et al. [18] proposed an RDHEI method to embed secret data in the encrypted JPEG bitstream. In [19] and [20], homomorphic encryption is utilized to encrypt the original image. However, image size increases because the used homomorphic encryption algorithm maps the pixel value into a larger data range. In above mentioned RDHEI methods, the content owner does nothing except for image encryption. These methods have a small payload and/or a high error rate in data extraction and image recovering.

To overcome these problems, some researchers aim to develop another type of separable RDHEI method by reserving the spare space for secret data embedding before image encryption. In Ma's method [21], it reserves the spare space by embedding some LSBs in a part of the cover image into the rest part of the cover image with using simplified RDH method in [22]. The self-embedding of LSBs ensures the reversibility of image recovering. Zhang et al. [23] selected some pixels and applied a histogram shifting method to their estimation error values for accommodating secret data.

Previous RDHEI methods in [12,13,15,14,21] have a limited embedding rate, and are under the only situation that the images are for the Cloud storage with no transmission involved and thus no attacks [18]. Considering the scenario that hospitals at different locations build a bridge for cooperations, many medical images embedded with patients' information or treatment history records will be shared among several working teams, thus medical images will be transmitted over public channels that they may inevitably experience some noise and data loss. In this scenario, these RDHEI methods may suffer from secret data loss when the marked encrypted image is partially damaged or lost. For example, method in [21] uses a part of the LSB plane in the encrypted image to accommodate the secret data when embedding rate is less than 0.2 bpp. If the LSB plane is illegally removed, all secret data will lose. In the separable method in [14], the secret data are embedded in the i^{th} most significant bit (MSB) plane, it will also suffer from complete loss of secret data when this MSB plane is removed or damaged.

To improve the embedding rate while enhancing security and robustness, this paper introduces the binary-block embedding (BBE) method to embed message bits in binary images. Based on BBE, we further propose a reversible data hiding algorithm in encrypted images (BBE-RDHEI). It first uses BBE to embed binary bits in several LSB planes of the original image into its MSB planes. BBE-RDHEI encrypts the original image and hides the secret data into its LSB planes. A bit-level scrambling process is then employed after secret data embedding to ensure that the proposed BBE-RDHEI can resist the noise and data loss attacks. Using different security keys, the receiver is able to obtain the secret data, marked decrypted image, decrypted image, or all of them from the marked encrypted image.

Our main contributions in this work are listed as follows:

(1) We propose a new BBE algorithm for reversible data hiding in the encryption domain, which is totally different from traditional RDH methods. BBE can be utilized in different types of images such as binary, gray-scale, medical and cartoon images.
(2) Based on BBE, we further propose a method of reversible data hiding in encrypted images, BBE-RDHEI. Compared with existing state-of-the-art methods, it has significantly improved embedding capacity and quality of the marked decrypted image. BBE-RDHEI can also be simplified and utilized for binary images, while existing RDHEI methods are designed only for gray-scale images.
(3) To significantly enhance the security level of BBE-RDHEI, we also propose a security key design mechanism such that BBE-RDHEI is able to resist the differential attack, while existing RDHEI methods cannot.
(4) To enhance the robustness of RDHEI methods in withstanding noise and data loss attacks, we introduce a bit-level scrambling process to BBE-RDHEI after secret data embedding to spread out embedded secret data over the entire marked encrypted image. As a result, BBE-RDHEI is able to recover most of secret data even if one bit-plane (e.g., LSB or MSB) of the marked encrypted image is completely removed. Moreover, any bit-level scrambling algorithm can be used in our BBE-RDHEI. This is another security benefit of BBE-RDHEI.

The rest of this paper is organized as follows: Section 2 will introduce the BBE algorithm. Section 3 will propose BBE-RDHEI. Simulation results and comparisons will be provided in Section 5. Section 6 will provide security and robustness analysis of the proposed BBE-RDHEI. Section 7 will draw a conclusion.

2. Binary-block embedding

In this section, we propose a binary-block embedding (BBE) algorithm to embed message bits into a binary image.

2.1. BBE

BBE first divides the binary image into a number of non-overlapping blocks, separates them into two groups named good and bad blocks, respectively. A good block is able to be embedded with messages while a bad one is not. In message embedding phase, BBE first labels the first 2 or 3 bits of each block with special bits that indicate the block types. Then the rest bits of a good block will be replaced with its structure information and message bits while the rest bits of a bad block will be kept unchanged. Next, we present the BBE algorithm in detail.

2.1.1. *Block labeling*

Assume that a binary image with a size of $M \times N$ is able to embed secret data. We first divide the image into a set of non-overlapping blocks with a size of $s_1 \times s_2$, where $s_1, s_2 \geq 3$. For a certain block, we let $n = s_1 * s_2$ be the total number of pixels within the block, and $m = \min\{n_0, n_1\}$ be the minimum value of n_0 and n_1, where n_0 and n_1 are the numbers of 0 s and 1 s within the block, respectively. According to a threshold n_a, we then classify these blocks into five categories as shown in Table 1, namely: Good-I/II/III/IV block and Bad block, where a good block is able embed secret data, while a bad one cannot.

Table 1
Block types and block-labeling bits.

Condition	Block type	Block description	Block-labeling bits
$m > n_a$	Bad	cannot embed data	00
$m = n_0 = 0$	Good-I	all pixels are 1	11
$m = n_1 = 0$	Good-II	all pixels are 0	10
$1 \leq m \leq n_a$, $n_0 < n_1$	Good-III	most of pixels are 1	011
$1 \leq m \leq n_a$, $n_1 < n_0$	Good-IV	most of pixels are 0	010

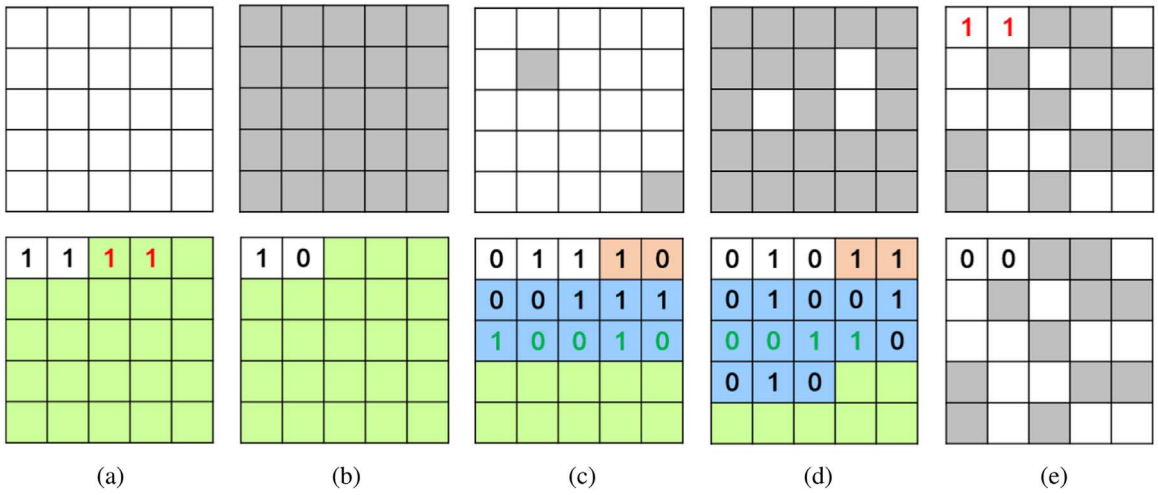

Fig. 1. BBE examples with the block size of 5×5. The first and second rows show the five types of original blocks and their corresponding embedded results. White and gray boxes represent the pixels with values of 1 and 0, respectively. The green, orange and blue areas are utilized to embed payload, parameter m and the positions of m original pixels in the block. (a) and (b) are Good-I and Good-II blocks with $c_b = 23$ and their embedded results; (c) Good-III block with $c_b = 10$ and its embedded result; (d) Good-IV block with $c_b = 7$ and its embedded result; and (e) Bad block with $c_b = -2$ and its result after embedding. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.).

In a Good-I (Good-II) block, all pixel values are equal to 1 (0), while in a Good-III (Good-IV) block, less than or equal to n_α pixel values are equal to 1 (0). An illustrative example can also be found in the first line of Fig. 1.

To embed the secret data, we first need to determine the block type. As shown in Table 1, Good-I (Good-II) block can be easily distinguished by checking the value of m, where m is obtained once a block is given. In order to distinguish Good-III (Good-IV) blocks from Bad ones, we calculate the threshold n_α by

$$n_a = \underset{x}{\operatorname{argmax}}\{n - 3 - \max\{\lceil \log_2 x \rceil, 1\} - x\lceil \log_2 n \rceil \geq 0\},$$
$$1 \leq x \leq \lceil 0.16{*}n \rceil \tag{1}$$

If $m \geq n_a$, it is a Bad block; otherwise, it is a Good-III (Good-IV) block. In Eq. (1), expressions $\max\{\lceil \log_2 x \rceil, 1\}$ and $x\lceil \log_2 n \rceil$ represent the bit length to store x and these x pixels' locations, respectively. These two parts are utilized to store the structure information of Good-III and Good-IV blocks, and they will be discussed in Section 2.1.2. Thus, after the block is labeled by 3 labeling bits and embedded with the structure information of x pixels, if there is still larger than or equal to 0 bits left, the current block is considered as a good block; otherwise, it is a bad block. Therefore, n_α is the maximum number of pixels that can be represented by the block itself with a given block size. It is utilized to distinguish bad blocks from good ones. Here, x should be less than n, and the reason why we set x less than or equal to $\lceil 0.16{*}n \rceil$ will be discussed in Section 2.4.1.

After determining the block type, we label each block by replacing its first 2 or 3 pixels with the corresponding block-labeling bits as shown in Table 1. Before labeling blocks, the first 2 or 3 original pixels of each block are picked up and stored with secret message for the purpose of image recovery at the receiver side.

2.1.2. *Structure information embedding*

In BBE, a good block is self-embedded with its original structure information and may have an additional spare space to accommodate secret message bits. The first 2 bits of each bad block are extracted and embedded into good blocks together with secret messages because they are directly replaced by the block-labeling bits '00' after the block labeling procedure.

For a Good-I (Good-II) block, no additional structure information needs to be embedded except for two labeling bits. For a Good-III (Good-IV) block, parameter m and the locations of m pixels need to be embedded as the structure information. Here, p bits are utilized to embed parameter m, where $p = \max\{\lceil \log_2 n_a \rceil, 1\}$. Then, we use variable length of bits to store the locations of m pixels. We first scan pixels in a block from top to bottom and left to right to obtain the location index values $\{z_i\}_{i=1}^m (1 \leq z_i \leq n)$ of these m pixels. For the first of m pixels located at z_1, without any additional information, z_1 could be any integer in the range of $[1, n]$. Thus, we use $\lceil \log_2 n \rceil$ bits to store its location index. For the second pixel located at z_2, it could only be in the range of $[z_1 + 1, n]$. Thus, we can use $\lceil \log_2(n - z_1) \rceil$ instead of $\lceil \log_2 n \rceil$ bits to store its location index. Therefore, the actual location information of m pixels are stored as the distance $\{t_i\}_{i=1}^m$ between the current pixel and its previous pixel, where t_i is calculated by Eq. (3). For example, for the second pixel, we store its location information by converting the decimal value t_2 into $\lceil \log_2(n - z_1) \rceil$-bit binary seqence. In this manner that considers the relative distance between adjacent pixels, we are able to use less bits to store the locations of m pixels. We then continue in this process until all m pixels' locations are stored. Thus, q_i bits are required to store the i^{th} pixel's location, and totally $(p + \sum_{i=1}^m q_i)$ bits are needed to store the parameter m and m pixels' locations.

$$q_i = \begin{cases} \lceil \log_2 n \rceil & \text{for } i = 1 \\ \max\{\lceil \log_2(n - z_{i-1}) \rceil, 1\} & \text{for } 2 \leq i \leq m \end{cases} \tag{2}$$

$$t_i = \begin{cases} z_i & \text{for } i = 1 \\ z_i - z_{i-1} & \text{for } 2 \leq i \leq m \end{cases} \tag{3}$$

2.1.3. *Message embedding*

The BBE payload $\mathcal{P}$ consists of two parts: $\mathcal{B}$ and $\mathcal{M}$, where $\mathcal{B}$ is a bit sequence containing all of the first 2 original bits in each bad block and $\mathcal{M}$ denotes secret messages. After embedding structure information, we replace the rest bits of a good block with c_b bits of payload $\mathcal{P}$, where c_b is the block capacity and calculated by

$$c_b = \begin{cases} n - 2 & \text{for } m = 0 \\ n - 3 - p - \sum_{i=1}^m q_i & \text{for } 1 \leq m \leq n_a \\ -2 & \text{otherwise} \end{cases} \tag{4}$$

The detailed procedures of BBE are provided in Algorithm 1.

Algorithm 1. Binary-Block Embedding.

Input: Binary image I, block size $s_1 \times s_2$, payload $\mathcal{P}$.
1: Divide I into non-overlapping blocks $B_{i,j}$ with a size of $s_1 \times s_2$. Calculate parameters n_a, p and q_i using Eqs. (1) and (2).
2: **for** each block $B_{i,j}$ **do**
3: Calculate n_0, n_1, m and c_b according to Eq. (4).
4: **if** $m > n_a$
5: Set the first two pixels to [0, 0].
6: **else if** m=0 and $n_0 = 0$ **then**
7: Set the first two pixels to [1, 1], replace other pixels by c_b bits of the payload.
8: **else if** m=0 and $n_1 = 0$
9: Set the first two pixels to [1, 0], replace other pixels by c_b bits of the payload.
10: **else if** $1 \le m \le n_a$ **then**
11: **if** $n_0 < n_1$ **then**
12: Set the first three pixels to [0, 1, 1].
13: **else**
14: Set the first three pixels to [0, 1, 0].
15: **end if**
16: Replace other pixels with the parameter m, locations of m pixels, and c_b bits of the payload.
17: **end if**
18: **end for**
Output: Embedded image $\hat{I}$.

Fig. 1 illustrates an example of BBE. A binary image is divided into 5 blocks with size of 5×5. The payload $\mathcal{P}$ includes the first two pixels of the bad block in Fig. 1(e) and 61 bits of secret messages. BBE embeds payload $\mathcal{P}$ into all good blocks one by one. For the block in Fig. 1(d), it is a Good-IV block where most pixels are 0 s. Parameters are m=3, p=2, $q_1 = 5$, $q_2 = q_3 = 4$, $c_b = 7$. BBE labels the block in Fig. 1(d) by setting its first three pixels to '010' (the white area), embeds $m = 3 = (11)_2$ to the subsequent two pixels (the orange area), puts the location of three pixels (white boxes in the original block in Fig. 1(d)) $t_1 = z_1 = 9 = (01001)_2$, $t_2 = z_2 - z_1 = 12 - 9 = 3 = (0011)_2$ and $t_3 = z_3 - z_2 = 14 - 12 = 2 = (0010)_2$ to the following 13 pixels (the blue region). The remaining 7 pixels (the green area) are utilized to embed payload $\mathcal{P}$. For the bad block in Fig. 1(e), its first two original pixels '11' are embedded at the beginning of the green area in Fig. 1(a). BBE replaces its first two pixels as '00' to indicate that it is a bad block, and keeps its other pixels unchanged.

2.2. Message extraction and image recovering

The message extraction and image recovering includes two phases: 1) payload extraction and good block recovering; 2) bad block recovering. In Phase 1, the BBE scans the first 3 labeling bits of each block to determine the block type. For a good block, BBE extracts parameter m, the locations of m pixels and payload bits from the block, and then reconstructs the block based on the extracted information. Otherwise, for a bad block, BBE records the block index. In Phase 2, BBE recovers the first 2 pixels of each bad block using the extracted payload and keeps the rest pixels unchanged.

2.2.1. *Phase 1*

For each image block, we first determine its block type by checking its first 3 pixels. If it is a bad block, we do nothing except for recording its block index. If it is a Good-I (Good-II) block, we obtain the payload bits from the last $(n-2)$ bits and recover the block by setting all bits to 1 s (0 s for the Good-II block). For a Good-III (Good-IV) block, we first extract the labeling bits and structure information from its first $(n - \hat{c}_b)$ pixels, where $\hat{c}_b$ is the block capacity; and then obtain the payload bits from the rest pixels of the block.

To extract secret data from a Good-III (Good-IV) block, we first obtain the raster-scanned bit sequence $[a_1, a_2, \ldots, a_n]$ from the block, and calculate parameter m from the specific p bits $[a_4, \ldots, a_{3+p}]$. Then, we calculate the location index distance $\{\hat{t}_i\}_{i=1}^{m}$ of m pixels by the following bits in an orderly way. Here, $\hat{t}_i$ is the location index distance between the i^{th} and $(i-1)^{th}$ pixels, and it is sequentially extracted from the subsequent $\{\hat{q}_i\}_{i=1}^{m}$ bits, where $\hat{q}_i$ is calculated by Eqs. (5). After obtaining $\hat{t}_i$, we then calculate the actual locations of m pixels by Eq. (6). For example, for the first pixel, we obtain its location index $\hat{z}_1 = \hat{t}_1$ according to the following $\hat{q}_1 = \lceil \log_2 n \rceil$ bits. For the second pixel, because its location can only in the range of $[\hat{t}_1 + 1, n]$, the maximum possible bits to store its location index should be $\hat{q}_2 = \lceil \log_2(n - \hat{t}_1) \rceil$. Thus, we obtain the distance $\hat{t}_2$ between the first and second pixels from the subsequent $\hat{q}_2$ bits and calculate the actual location $\hat{z}_2 = \hat{t}_1 + \hat{t}_2$ of the second pixel. We continue in this manner until all structure information of m pixels are successfully obtained.

$$\hat{q}_i = \begin{cases} \lceil \log_2 n \rceil & \text{for } i = 1 \\ \max\{\lceil \log_2(n - \hat{z}_j) \rceil, 1\} & \text{for} 2 \le i \le m \end{cases} \quad (5)$$

$$\hat{z}_i = \begin{cases} \hat{t}_i & \text{for } i = 1 \\ \hat{t}_{i-1} + \hat{t}_i & \text{for } 2 \le i \le m \end{cases} \quad (6)$$

Finally, we extract the payload bits from the rest $\hat{c}_b$ pixels of the block and set all pixels to 1 s (0 s for the Good-IV block) except for the m pixels with the index values $\{\hat{z}_i\}_{i=1}^{m}$, where $\hat{c}_b$ is calculated by

$$\hat{c}_b = \begin{cases} n - 2 & \text{for } m = 0 \\ n - 3 - p - \sum_{i=1}^{m} \hat{q}_i & \text{for } 1 \le m \le n_a \\ -2 & \text{otherwise} \end{cases} \quad (7)$$

2.2.2. *Phase 2*

After obtaining the extracted payload and bad block indices, we recover the first two pixels of each bad block using two bits of the payload. Thus, the remaining payload bits are extracted messages.

The procedures of message extraction and image recovering of BBE are given in Algorithm 2.

Algorithm 2. Message extraction and image recovering.

Input: Image $\hat{I}$ with messages, block size $s_1 \times s_2$.
1: Initialization: $\mathcal{P} = [\ ]$, bad block index $b = [\ ]$.
2: Divide $\hat{I}$ into non-overlapping blocks $\hat{B}_{i,j}$ with a size of $s_1 \times s_2$. Calculate parameters n and p.
3: **for** each block $\hat{B}_{i,j}$ **do**
4: Scan $\hat{B}_{i,j}$ to obtain the pixel sequence $[a_1, a_2, a_3, \ldots, a_n]$.
5: **if** $[a_1, a_2] = [1, 1]$ **then**
6: $\mathcal{P} \leftarrow [\mathcal{P}, [a_3, a_4, \ldots, a_n]]$. $\hat{B}_{i,j} \leftarrow$ Set all pixels to 1.
7: **else if** $[a_1, a_2] = [1, 0]$ **then**
8: $\mathcal{P} \leftarrow [\mathcal{P}, [a_3, a_4, \ldots, a_n]]$. $\hat{B}_{i,j} \leftarrow$ Set all pixels to 0.
9: **else if** $[a_1, a_2] = [0, 1]$ **then**
10: Calculate m, obtain the m pixels' location $\hat{z}_i$ using Eq. (6).
11: $\mathcal{P} \leftarrow [\mathcal{P}, [a_{4+p+\sum_{i=1}^{m} q_i}, \ldots, a_n]]$.
12: **if** $a_3 = 1$ **then**
13: $\hat{B}_{i,j} \leftarrow$ Set all pixels to 1 except for the m pixels with location $\hat{z}_i$.
14: **else**
15: $\hat{B}_{i,j} \leftarrow$ Set all pixels to 0 except for the m pixels with location $\hat{z}_i$.
16: **end if**
17: **else**
18: $b \leftarrow [b; (i, j)]$. % record the bad block index
19: **end if**

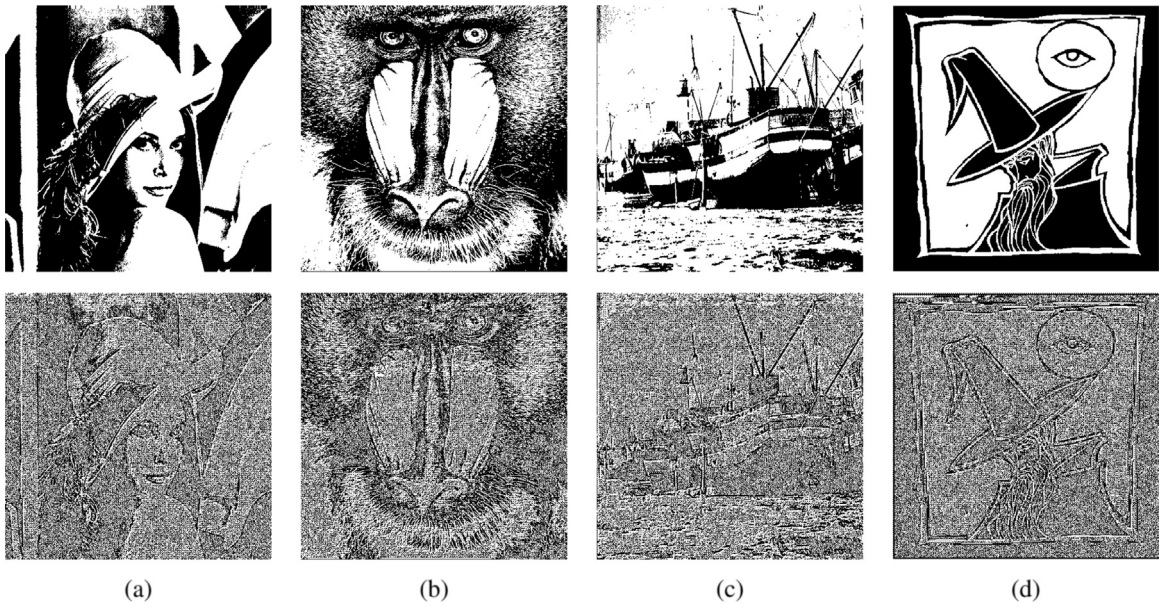

Fig. 2. Embedding results of BBE with a block size $s_1 = s_2 = 4$. The first and second rows show the original images and their embedded results with an embedding rate (a) 0.7100 bpp, (b) 0.3536 bpp, (c) 0.6332 bpp and (d) 0.6730 bpp.

20: **end for**
21: Extract $\mathcal{B}$ and $\mathcal{M}$ from $\mathcal{P}$.
22: **for** each bad block $\widehat{B}_{i,j}$ where $(i, j) \in b$ **do**
23: Replace the first 2 pixels of $\widehat{B}_{i,j}$ by 2 bits of $\mathcal{B}$.
24: **end for**
Output: Recovered image I, message $\mathcal{M}$.

2.3. Simulation results

BBE introduces noise to uniform regions, changes the uniform regions, and keeps only the edges. These operations yield an image with more noise. Fig. 2 shows the embedding results of four 512×512 binary images using the BBE algorithm with the block size of 4×4. As we can obverse, the more all-white or all-black blocks the original image contains, the higher embedding rate BBE can achieve. Meanwhile, the embedded images become more noise-like.

2.4. Discussions

Here, we discuss the threshold n_a, embedding rate and advantages of BBE.

2.4.1. Threshold n_a

As shown in Eq. (1) and Table 1, the block size $s_1 \times s_2$ will influence the value of n_a, which is a threshold to decide the block types. Fig. 3 visually shows the relationship between the threshold n_a and block pixel number n. We set $3 \le s_1, s_2 \le 40$, thus, $n \in [9, 1600]$. As can be seen, n_a increases with the increase of n. For example, if $s_1 = s_2 = 3$, we can obtain the threshold n_a=1. This means that a 3×3 image block containing at most one 0 or 1 is considered as a good block, otherwise it is a bad block. In addition, when block size is 5×5, n_a/n reaches the maximum value of 0.16. Therefore, in Eq. (1), x should be less than or equal to $\lfloor 0.16*n \rfloor$.

2.4.2. Embedding rate

To analyze the embedding rate, we apply BBE with different block sizes to 10,000 binary test images that are generated by binarizing the gray-scale images from BOWSBase[1] with the threshold calculated by Otsu's method [24]. Table 2 lists the average embedding rates of all test images with different block sizes. Here we set $s_1 = s_2$. From the results, we can observe that, the average embedding rate reaches the maximum value when the block size is 7×7. When block size is less than 7×7, the average embedding rate increases with the block size enlarging, and it decreases when block size is larger than 7×7.

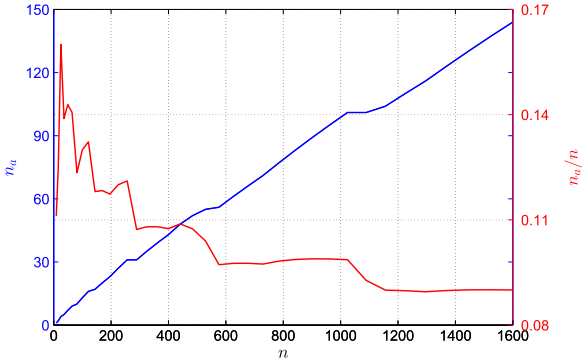

Fig. 3. Relationship between threshold n_a and block pixel number n.

Table 2
Average embedding rate of 10,000 binary images using BBE with different block sizes.

Block size $s_1 \times s_2$	Average embedding rate (.bpp)	Block size $s_1 \times s_2$	Average embedding rate (.bpp)
3×3	0.6628	11×11	0.7308
4×4	0.7431	14×14	0.7014
5×5	0.7600	17×17	0.6975
6×6	0.7628	21×21	0.6568
7×7	0.7685	28×28	0.6180
8×8	0.7679	33×33	0.5708
9×9	0.7348	40×40	0.5072

2.4.3. Advantages

The proposed BBE has at least the following advantages. Namely, BBE is able to

(a) achieve a higher embedding rate when the original image contains

[1] The BOWSBase database is located in: http://bows2.ec-lille.fr/.

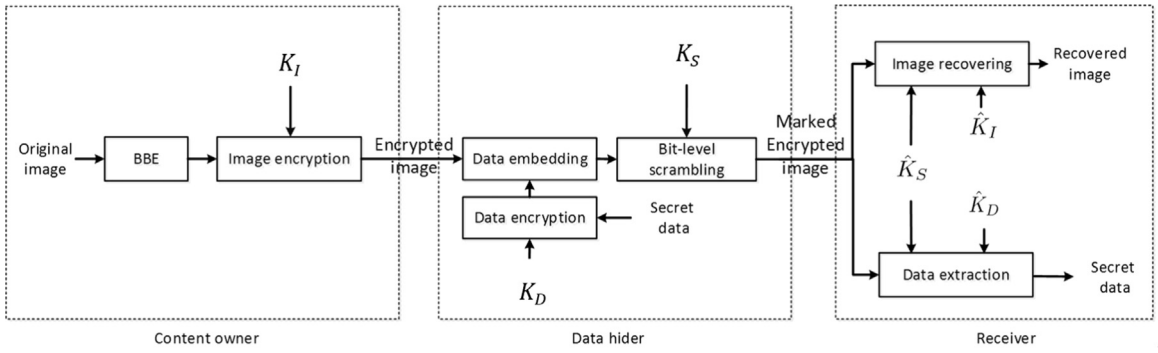

Fig. 4. The structure of BBE-RDHEI.

more all-white or all-black blocks.

(b) perform data hiding and image quality degradation within one single step. This is because BBE embeds secret messages by modifying the pixel values in good blocks while keeping the randomness of bad blocks.

(c) completely recover the secret messages and original image without any error.

3. BBE based reversible data hiding in encrypted images

In this section, we propose a BBE based reversible data hiding algorithm for encrypted images (BBE-RDHEI). The structure of BBE-RDHEI is shown in Fig. 4. It is composed of three processes: generation of the encrypted image, generation of the marked encrypted image, data extraction/image recovery. These processes are accomplished by the content owner who provides the original image, the data hider who has the secret data to be embedded and the receiver, respectively. The content owner uses the BBE algorithm to embed binary bits of lower bit-planes of the original image into its higher bit-planes such that its lower bit-planes can be reserved for hiding secret data in the subsequent processes. The image is then encrypted using the image encryption key K_I. The data hider encrypts the secret data using the data encryption key K_D, embeds them into the reserved lower bit-planes in the encrypted image, and scrambles the image using the sharing encryption key K_S to generate the marked encrypted image which will be transmitted over public channels. These three encryption keys are randomly generated by users.

3.1. Random sequence generation

Before presenting three processes of BBE-RDHEI in detail, we discuss the security key design and random sequence generation. Their framework is shown in Fig. 5.

A secure hash algorithm[2] (SHA) is used to generate two random hash sequences with the inputs of a user-defined security key K and image/secret data, respectively. Then the two hash sequences are XORed to generate the inner random sequence $\widehat{K}$. $\widehat{K}$ is utilized to initialize a chaotic system to produce the random sequence that will be used for encrypting the original image and secret data. The random sequence $\widehat{K}$ is useful for secret data extraction and image recovering. Thus, it is called the decryption key. The length of security key K is user-defined and the decryption key $\widehat{K}$ is with the same length of the output of SHA.

Using the framework in Fig. 5, any change in the image/secret data or encryption key will result in the totally different decryption key and random sequence. This enhances the security performance of the proposed BBE-RDHEI.

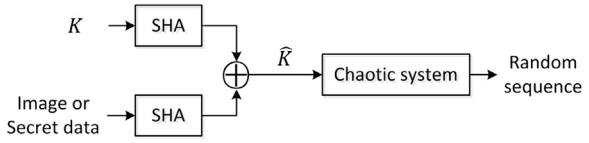

Fig. 5. The generation framework of the security key and random sequence.

3.1.1. Security key design

BBE-RDHEI has three encryption keys, K_I, K_D and K_S. They all are random bit sequences generated by users. In addition to obtaining the marked encrypted image, BBE-RDHEI also produces three corresponding decryption keys, $\widehat{K}_I$, $\widehat{K}_D$ and $\widehat{K}_S$, for the receiver to extract the secret data, marked decrypted image, decrypted image, or all of them if he holds $\widehat{K}_S$ along with $\widehat{K}_D$, $\widehat{K}_I$ or both, respectively. These decryption keys are linked with their corresponding encryption keys and contents of images or secret data as shown in Fig. 5. They are defined by

$$\begin{bmatrix} \widehat{K}_I \\ \widehat{K}_D \\ \widehat{K}_S \end{bmatrix} = \begin{bmatrix} \mathbb{H}(I) \\ \mathbb{H}(P) \\ \mathbb{H}(E) \end{bmatrix} \oplus \begin{bmatrix} \mathbb{H}(K_I) \\ \mathbb{H}(K_D) \\ \mathbb{H}(K_S) \end{bmatrix} \tag{8}$$

where $\mathbb{H}(.)$ is an SHA; I, P and E represent the original image, secret data, and encrypted image, respectively.

Note that users have flexibility to choose any SHA for Eq. (8). In this paper, we select SHA-1[3] for simulations. Thus, three decryption keys and outputs of $\mathbb{H}(.)$ have a length of 160 bits. According to Eq. (8) $\widehat{K}_I$ is linked to K_I and the original image I; $\widehat{K}_D$ is linked to K_D and the secret data P; $\widehat{K}_S$ is linked to K_S and the encrypted image E. Thus, any change in the encryption key or the input of $\mathbb{H}(.)$ will result in a completely different decryption key and thus another chaotic sequence.

3.1.2. Chaotic sequence generation

A chaotic sequence is a random sequence that is sensitive to the parameter and initial value of its chaotic system. Any chaotic system can be used to generate the chaotic sequence, and we choose the Logistic-Sine system (LSS) [25] for demonstrations and it is defined by

$$x_{i+1} = (yx_i(1 - x_i)) + (4 - y)sin(\pi x_i)/4) \bmod 1 \tag{9}$$

where the initial value x_0 ($x_0 \in [0, 1]$) and parameter y ($y \in (0, 4]$) are calculated by Algorithm 3 with a binary hash sequence H. Then, we will use this initial condition (x_0, y) in rest of this paper.

Algorithm 3. Generation of initial value and parameter of LSS.

Input: Binary sequence
$H = [h_1, h_2, \ldots, h_{160}](h_i \in \{0, 1\}, 1 \le i \le 160)$.

1: $u_1 \leftarrow \sum_{i=1}^{40} h_i 2^{40-i}$

2: $u_2 \leftarrow \sum_{i=41}^{80} h_i 2^{80-i}$

3: $v_1 \leftarrow \sum_{i=81}^{120} h_i 2^{120-i}$

[2] http://en.wikipedia.org/wiki/Secure_Hash_Algorithm.

[3] http://tools.ietf.org/html/rfc3174.

4: $v_2 \leftarrow \sum_{i=121}^{160} h_i 2^{160-i}$
5: **Initial value** $x_0 \leftarrow u_1/2^{40}$
6: Parameter $y_0 \leftarrow u_2/2^{40}$
7: **for** i=1 to 2 **do**
8: $x_i \leftarrow (x_{i-1} u_i v_i/2^{40} + x_{i-1}) \bmod 1$
9: $y_i \leftarrow (y_{i-1} u_i v_i/2^{40} + y_{i-1}) \bmod 4$
10: **end for**
11: $x_0 \leftarrow x_2$
12: $y \leftarrow 4 - y_2$
Output: Initial conditions (x_0, y).

In the following subsections, we will discuss three processes of BBE-RDHEI one by one.

3.2. Generation of the encrypted image

To protect the content of the original image while embedding secret data, the content owner applies BBE to the original image to reserve the bit space for data hiding, encrypts the image, and then sends it to the data hider. Next, we present these steps in detail.

3.2.1. Reserving bit space using BBE

Reserving the bit space is to embed binary bits of LSB planes of the original image into its MSB planes using BBE in order to reserve these LSB planes for embedding secret data.

An original image I with a size of $M \times N$ and a data range of $[0, 255]$ is decomposed into 8 bit-planes. Each bit-plane is then divided into a set of $s_1 \times s_2$ blocks. For simplicity, we set $s_1 = s_2 = s$. We calculate the capacity of each block using Eq. (4) and add them together to obtain the capacity of each bit-plane C_i $(1 \le i \le 8)$, where C_1 and C_8 are the MSB and LSB planes of the original image I, respectively. Suppose k MSB planes are able to embed C bits of the LSBs that are selected from other $(8 - k)$ LSB planes. Here C is the image capacity calculated by

$$C = \begin{cases} 0 & \text{if } k = 0 \\ \sum_{i=1}^{k} C_i & \text{otherwise} \end{cases} \tag{10}$$

where

$$k = \begin{cases} 0 & \text{if } C_1 \le 0 \\ \arg\max_t \{ \frac{\sum_{i=1}^{t} C_i}{MN} \le 8 - t, t = 1, \ldots, 8 \} & \text{if } C_i > 0 \end{cases} \tag{11}$$

If C=0, the original image is unable to embed secret data. Otherwise, the algorithm uses BBE to construct and embed the payload $\mathcal{P}$ into k MSB planes of the original image. Here $\mathcal{P}$ is composed of bit sequences $\mathcal{B}$ and $\mathcal{M}$, where $\mathcal{B}$ contains all the first two bits of each bad block in k MSB planes and $\mathcal{M}$ includes C binary bits that are selected sequentially from LSB planes of the original image. After embedding $\mathcal{P}$ to k MSB planes, C bit positions in LSB planes are reserved for hiding secret data.

3.2.2. Image encryption

After reserving the bit space using BBE, the content owner encrypts the resulting image R to obtain the encrypted image E using bit-level XOR operation by

$$E(i, j) = R(i, j) \oplus T(i, j) \tag{12}$$

where (i,j) represent the pixel location. T is an $M \times N$ substitution matrix that is reshaped from a vector $\mathcal{T} = [t_1, t_2, \ldots, t_{MN}]$, where t_i $(1 \le i \le MN)$ is calculated by

$$t_i = \lfloor x_i \times 2^{50} \rfloor \bmod 256 \tag{13}$$

and x_i is generated from Eq. (9), where the initial condition (x_0, y) is generated from Algorithm 3 with $\widehat{K}_I$, where $\widehat{K}_I$ is generated by Eq. (8) using the original image I and image encryption key K_I. The output of LSS randomly changes in the entire data range of $[0, 1]$ as long as the initial value x_0 and parameter y are in the ranges of $[0, 1]$ and $(0, 4]$, respectively. LSS has good chaotic performance in the whole parameter ranges as proved in [25]. Thus, any parameter generated from Algorithm 3 can be selected to initialize LSS.

Finally, the content owner converts the capacity C into a 20-bit binary sequence and embeds it into LSBs of the first 20 pixels of image E using bit replacement. This is used to tell the data hider how many secret bits can be embedded into the encrypted image E. Thus, the maximum data embedding rate r (.bpp) can be calculated by

$$r = C/(MN) \tag{14}$$

The parameters k and s need to send to receiver together with decryption key $\widehat{K}_I$ for image recovering.

3.3. Generation of the marked encrypted image

After obtaining the encrypted image E, the data hider embeds secret data into E without knowing the image content, and then scrambles the image with the sharing key K_S to generate the marked encrypted image.

3.3.1. Data embedding

Because BBE has reserved the bit space in LSB planes for data hiding, the data hider can embed secret data into LSB planes using bit replacement with the information provided by LSBs of the first 20 pixels. In order to achieve a higher level of security, the data hider encrypts the secret data bits $P = [p_1, p_2, \ldots, p_u] (1 \le u \le C)$ using the data encryption key K_D and embeds the encrypted secret data $\widehat{P} = [\widehat{p}_1, \widehat{p}_2, \ldots, \widehat{p}_u]$ into image E.

$$\widehat{p}_i = \text{round}(x_i) \oplus p_i \ (1 \le i \le u) \tag{15}$$

x_i is calculated from Eq. (9), the initial condition (x_0, y) is calculated from Algorithm 3 using the data hiding key $\widehat{K}_D$, where $\widehat{K}_D$ is generated by Eq. (8) with the secret data P and data encryption key K_D.

The final encrypted image with embedded secret data is denoted by D. The bit length u of secret data needs to send to receiver as a part of the data decryption key $\{\widehat{K}_D, u\}$ for secret data extraction.

3.3.2. Bit-level scrambling

After obtaining the image D embedded with secret data, the data hider further scrambles D with the sharing key K_S to obtain the final scrambled image with hidden secret data that is called the marked encrypted image $\hat{I}$. Any bit-level scrambling/permutation method can be used to scramble the image. We combine the permutation method in [26] with LSS in Eq. (9) for our experiments. The initial condition (x_0, y) is calculated by Algorithm 3 with $\widehat{K}_S$, where $\widehat{K}_S$ is generated by Eq. (8) using the encrypted image E and sharing encryption key K_S. The secret data length u needs to send to receiver as a part of the sharing decryption key $\{\widehat{K}_S, u\}$ for secret data extraction and image recovering.

3.4. Data extraction and image recovering

The data extraction and image recovering are two independent processes for the receiver. Using different security keys, he can either recover the image, extract secret data, or obtain both.

3.4.1. Data extraction

Using image and sharing decryption keys, the receiver can extract the secret data without knowing the image content. BBE-RDHEI first unscrambles the marked encrypted image $\hat{I}$ using $\widehat{K}_S$ to obtain the unscrambled image $\widehat{D}$. Then it extracts the encrypted secret data $\widehat{P} = [\widehat{p}_1, \widehat{p}_2, \ldots, \widehat{p}_u]$ from LSB planes, and decrypts $\widehat{P}$ using $\widehat{K}_D$ to obtain the original secret data $P = [p_1, p_2, \ldots, p_u]$ by

$$p_i = \text{round}\,(x_i) \oplus \hat{p}_i \ (1 \le i \le u) \tag{16}$$

where x_i is generated by Eq. (9), where the initial condition (x_0, y) is calculated from Algorithm 3 with $\hat{K}_D$.

3.4.2. Image recovering

If the receiver has the decryption keys $\hat{K}_S$ and $\hat{K}_I$ with the parameters k, s, and u, he can obtain the original image or the marked decrypted image that is similar to the original image embedded with the secret data.

To generate the marked decrypted image, BBE-RDHEI first obtains the unscrambled image $\hat{D}$, decrypts the image by

$$\hat{R}\,(i, j) = \hat{D}\,(i, j) \oplus T\,(i, j) \tag{17}$$

except for the secret data bits, where T is the substitution matrix generated by Eq. (13), where the initial condition (x_0, y) is calculated by Algorithm 3 with $\hat{K}_I$. Then, BBE-RDHEI extracts LSBs from k MSB planes using Algorithm 2, and recovers the MSB planes of the marked decrypted image. If the receiver also holds $\hat{K}_D$, he can further extract and decrypt the secret data from the marked decrypted image using Eq. (16).

To completely recover the original image from the marked decrypted image, BBE-RDHEI further replaces the secret data bits using binary bits extracted from the k MSB planes. Because of the reversibility of BBE, the image can be completely recovered without any error.

4. Discussion

In this section, we discuss the security keys and parameters used in the proposed BBE-RDHEI, and the advantages of BBE-RDHEI.

4.1. Security keys

There are six security keys used in the proposed BBE-RDHEI, namely the image encryption and decryption keys, data encryption and decryption keys, and sharing encryption and decryption keys. Table 3 gives a list of all security keys. At the receiver side, the sharing decryption key is combined with the data or image decryption key for data extraction or image recovering, respectively. For example, decryption keys $\{\hat{K}_D, \hat{K}_S, u\}$ are for data extraction and $\{\hat{K}_I, \hat{K}_S, k, s, u\}$ are utilized for image recovering. In BBE-RDHEI, we reserve the spare space by embedding the binary bits in LSB planes of the original image into its MSB planes, and the encrypted secret data is then embedded into the reserved LSB planes. The sharing keys and bit-level scrambling procedure are designed to ensure that these secret data are extremely difficult to be located and that most of secret data can be recovered when a part of the LSB or MSB planes of the marked encrypted image are lost during transmission. This will be verified by experiments in Section 6.

4.2. Parameters

There are five parameters used in the proposed BBE-RDHEI: the block size s, number of bitplanes k that are used to embed binary bits in LSB planes, secret data length u, and the initial condition (x_0, y) that is utilized to initialize the LSS. Generally, s is user-defined. How the value of s influences the maximum embedding rate will be discussed in Table 4 in Section 5. Given an original image and block size s, the number of bitplanes k is fixed and can be calculated by Eq. (11). The secret data length u is also fixed when the secret data is given. Because the LSS has good chaotic performance in its whole data ranges of [0, 1] (for x_0) and (0, 4] (for y), the initial condition (x_0, y) will randomly distribute in their corresponding data ranges. This will be also verified by experiments in Fig. 10 in Section 6.

Table 3
List of security keys.

Encryption keys	Notation	Decryption keys	Notation
Image encryption key	$\{K_I, s\}$	Image decryption key	$\{\hat{K}_I, k, s\}$
Data encryption key	K_D	Data decryption key	$\{\hat{K}_D, u\}$
Sharing encryption key	K_S	Sharing decryption key	$\{\hat{K}_S, u\}$

Table 4
Average embedding rate of 10,000 grayscale images using BBE-RDHEI with different block sizes.

Block size $s \times s$	Average embedding rate r (.bpp)	Block size $s \times s$	Average embedding rate r (.bpp)
3×3	1.9379	9×9	2.0490
4×4	2.1748	11×11	2.0116
5×5	2.2034	12×12	1.9466
6×6	2.1752	14×14	1.8867
7 × 7	2.1774	17×17	1.8310
8×8	2.1556	20×20	1.6874

4.3. Advantages of the proposed algorithm

In summary, the proposed BBE-RDHEI has at least following advantages.

- **Security:** Three security keys are being utilized to ensure different levels of the user's privileges. The receiver is able to extract different contents (the secret data or cover image) by using different combinations of security keys. Moreover, the generation scheme of the security keys ensures that any change in user-defined security keys or input contents of SHA in Eq. (8) (the original image or secret data) will result in a totally different output (the marked encrypted image). This allows BBE-RDHEI to resist the differential attack.
- **Reversibility:** The secret data and original image can be fully recovered due to the reversibility of BBE and independence of the data extraction and image recovering processes.
- **Extendability:** BBE-RDHEI can be simplified and utilized in binary images as well (e.g., we can embed a part of data in a binary image into the rest portion of the binary image to reserve spare space for secret data embedding.).

5. Simulation results and comparisons

The proposed BBE-RDHEI is implemented in Matlab. All test images in our experiments have a size of 512×512 and the pixel value range of [0, 255].

Fig. 6 shows the simulation results of BBE-RDHEI in the standard gray-scale Lena image with parameter s=8 and the embedding rate r=1.6834 bpp. As we can obverse, the marked encrypted image is a noise-like image. It protects both the original image and secret data. The unauthorized user has extremely difficulty to obtain any useful information from it. Two decrypted images (Figs. 6(c) and (d)) have no visual difference although the LSB planes of the image in Fig. 6(c) carry secret data.

Table 4 lists the average embedding rates of 10,000 images in BOWSBase[4] using BBE-RDHEI under different block sizes. As can be seen, when the block size s=5, the images have the maximum embedding rates. When s is larger than 5, the embedding rate decreases while the block size increases. Even s is as large as 20, BBE-RDHEI has an average embedding rate of 1.6874 bpp.

To compare the embedding rate, we apply BBE-RDHEI and five existing RDH methods to several selected images as shown in Fig. 7. The results are plotted in Fig. 8. In experiments, we set the block size

[4] The BOWSBase database is located in: http://bows2.ec-lille.fr/.

Fig. 6. Data hiding and extraction using BBE-RDHEI with the block size 8×8 and embedding rate r=1.6834 bpp. (a) The original image. (b) the marked encrypted image; (c) the marked decrypted image; (d) the decrypted image.

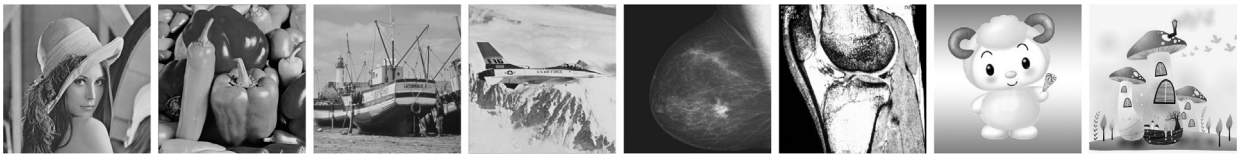

Fig. 7. Test images used in Fig. 8. (a) Lena; (b) Peppers; (c) Boat; (d) Airplane; (e) Breast; (f) Bone; (g) Cartoon1; (h) Cartoon2.

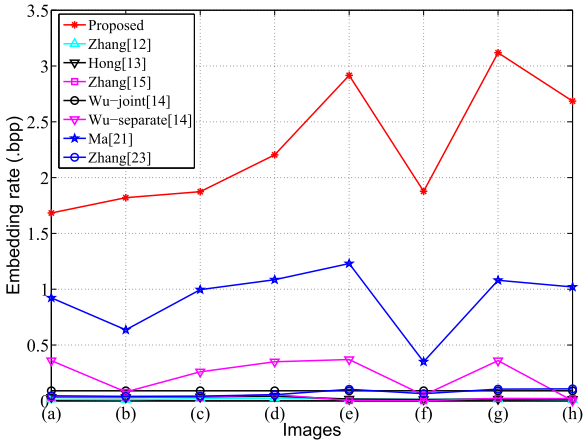

Fig. 8. Comparison of the embedding rate of different methods in several selected images. (a)-(h) are test images shown in Fig. 7.

5×5 for BBE-RDHEI, and 8×8 for Zhang's [12] and Hong's [13] methods. Because Zhang's [12] and Hong's [13] methods may not completely extract the secret data, the pure embedding rates are calculated by $(1-\mathcal{H}(\rho))/(s_1 s_2)$, where $\mathcal{H}(\rho)$ is the binary entropy function with the error rate ρ. For Zhang's methods in [15], the embedding rates are under the situation of full image recovery. For Wu's joint method in [14], we use 4 pixels in a group to embed 1 secret data bits by flipping the 6^{th} LSB of pixels within the group; and we choose the results with a high probability of successful data extraction. For Wu's separate method [14], we embed the secret data by modifying the MSB of the selected pixel, and we choose the results with full correctly image recovering. For Ma's method [21], the simplified RDH method in [22] is utilized for self-embedding. For Zhang's method in [23], we use 20% pixels for estimation. The results in Fig. 8 show that methods of reserving the spare space before encryption achieve much better data hiding performance than those without preprocessing before encryption. This is because a normal image contains more redundancy than in a noise-like encrypted image. Our proposed BBE-RDHEI performs well not only in gray-scale, medical and cartoon images, and has an embedding rate nearly two times more than these state-of-the-art RDH methods in most cases.

Fig. 9 compares visual quality of the marked decrypted images generated by BBE-RDHEI and several existing methods. We select a nature, medical and cartoon image from Fig. 7 for comparisons; separately. The results show that BBE-RDHEI has the highest PSNR scores in the marked decrypted images under different embedding rates. This is because BBE-RDHEI uses only LSBs for embedding secret data while existing methods require higher bit-planes for accommodating secret data.

6. Security and robustness analysis

When a marked encrypted image is transmitted through public channels, hackers may try to break it to obtain the secret data or/and original image; meanwhile, the marked encrypted image may inevitably experience some noise and data loss during transmission. Thus, an RDHEI algorithm should be able to withstand potential attacks. Modern steganalysis is to detect the existence of the hidden data in a meaningful stego image. Once the presence of hidden information is revealed or even suspected, the purpose of steganography is defeated, even if the secret data is not extracted or deciphered [27]. Different from the traditional steganographic methods, the proposed BBE-RDHEI as well as existing RDHEI methods [12,13,15,16,21] embeds secret data in an encrypted image (i.e., random-like image) and intends to protect both the original image and secret data from being illegally extracted or damaged. Therefore, in this section, we analyze the security of the proposed BBE-RDHEI in against the brute-force and differential attacks [28–30]. Moreover, because all public channels are noise channels, we also analyze the robustness of BBE-RDHEI in resisting the noise and data loss attacks.

6.1. Security analysis

6.1.1. Brute-force attack

The brute-force attack is a commonly used ciphertext-only attack. In this attack, the hackers exhaustively search all possible keys until the correct one is found. As mentioned in Sections 3, three security keys, $\widehat{K}_I$, $\widehat{K}_D$ and $\widehat{K}_S$, are needed to ensure the successful recovering of the secret data and original image. Obtaining either the original image or secret data needs at least two security keys. Each key is a length of 160 bits, therefore the possible combinations of the security keys will be more than 2^{320}. Thus the key space is sufficiently large to resist the brute-force attack.

On the other hand, because the security keys are to set the initial condition (x_0, y) of LSS for image encryption, a large data range of x_0

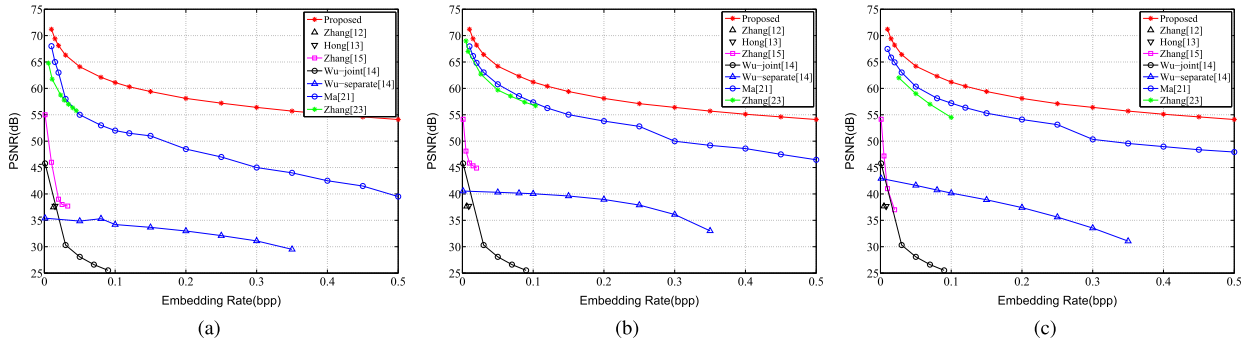

Fig. 9. PSNR comparison of the marked decrypted images generated by different RDHEI methods with different embedding rates. (a) Lena; (b) Breast; (c) Catoon1.

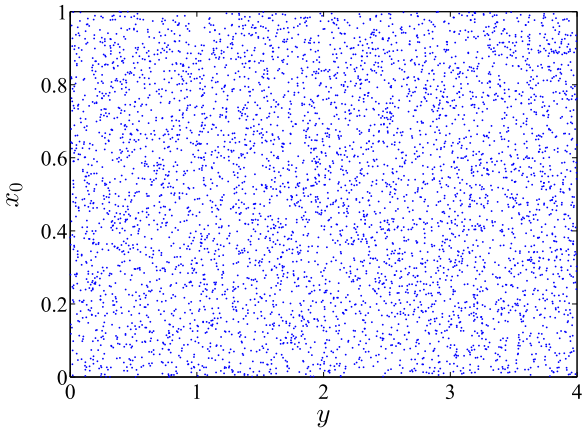

Fig. 10. Distribution of 5000 pairs of x_0 and y that are generated from Algorithm 3 with the input of 5000 randomly generated 160-bit binary sequences, separately.

and y will reduce their probability of being predicted and result in high security. In order to show the distribution of x_0 and y, we randomly generate 5000 160-bit binary sequences, and set them to the input of Algorithm 3 to generate 5000 pairs of initial condition (x_0, y), separately. Fig. 10 shows the distributions of the initial condition (x_0, y). We can obverse that the values of x_0 and y randomly distribute in the whole data ranges of [0, 1] and (0, 4], respectively. This demonstrates that it is extremely difficult for the hackers to obtain the correct initial condition (x_0, y) for decryption.

6.1.2. *Differential attack*

The differential attack is a kind of chosen-plaintext attack in which the hackers try to break the encrypted information by analyzing the correlation of changes between the input plaintexts and output ciphertexts [31]. Differential analysis is under the assumption that security keys are not changed during attack or analysis. To resist the differential attack, an algorithm should ensure that a tiny change in the inputs will result in a significant change in the outputs. For RDHEI, there are two types of plaintexts (the original image and secret data) and one ciphertext (the marked encrypted image). Thus, we perform differential analysis to the original image and secret data, separately.

To evaluate its performance of differential analysis to the original image, we apply BBE-RDHEI to two original images using the same secret data and security keys. The results are shown in Fig. 11. The original image in Fig. 11(b) is generated from the image in Fig. 11(a) by setting its pixel value 107 in position (455, 288) to 0. As one can

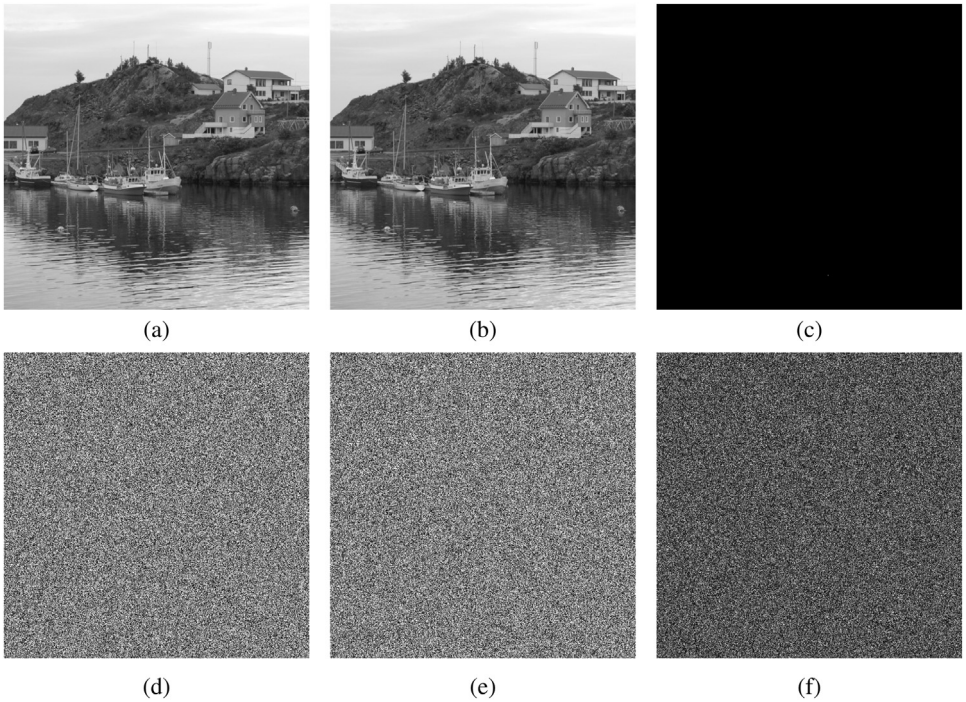

Fig. 11. Differential analysis of BBE-RDHEI for the original image with block size s=10, embedding rate r=1.6047 bpp. (a) The original image I_1; (b) the original image I_2, which is obtained from (a) with one pixel difference; (c) the difference between (a) and (b), $|I_1 - I_2|$; (d) the marked encrypted image $\hat{I}_1$ of (a); (e) the marked encrypted image $\hat{I}_2$ of (b); (f) the difference between (d) and (e), $|\hat{I}_1 - \hat{I}_2|$.

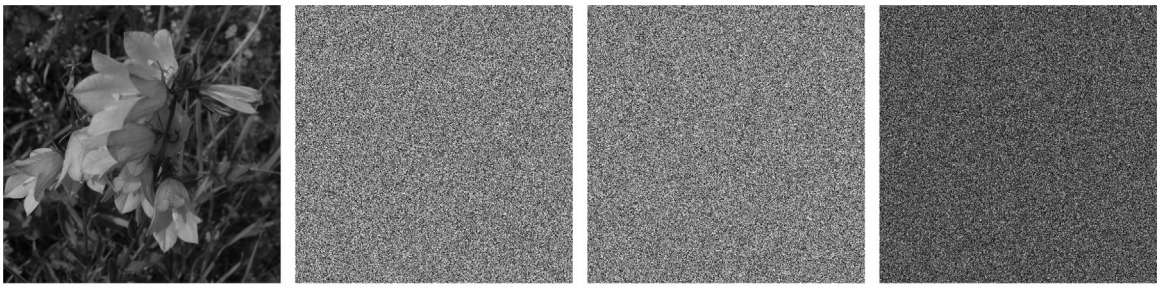

Fig. 12. Differential analysis of BBE-RDHEI for the secret data with block size s=16, embedding rate r=1.2750 bpp. (a) The original image; (b) the marked encrypted image embedded with message P_1; (c) the marked encrypted image embedded with message P_2 where P_1 and P_2 are only with one bit difference; (d) the difference between (b) and (c).

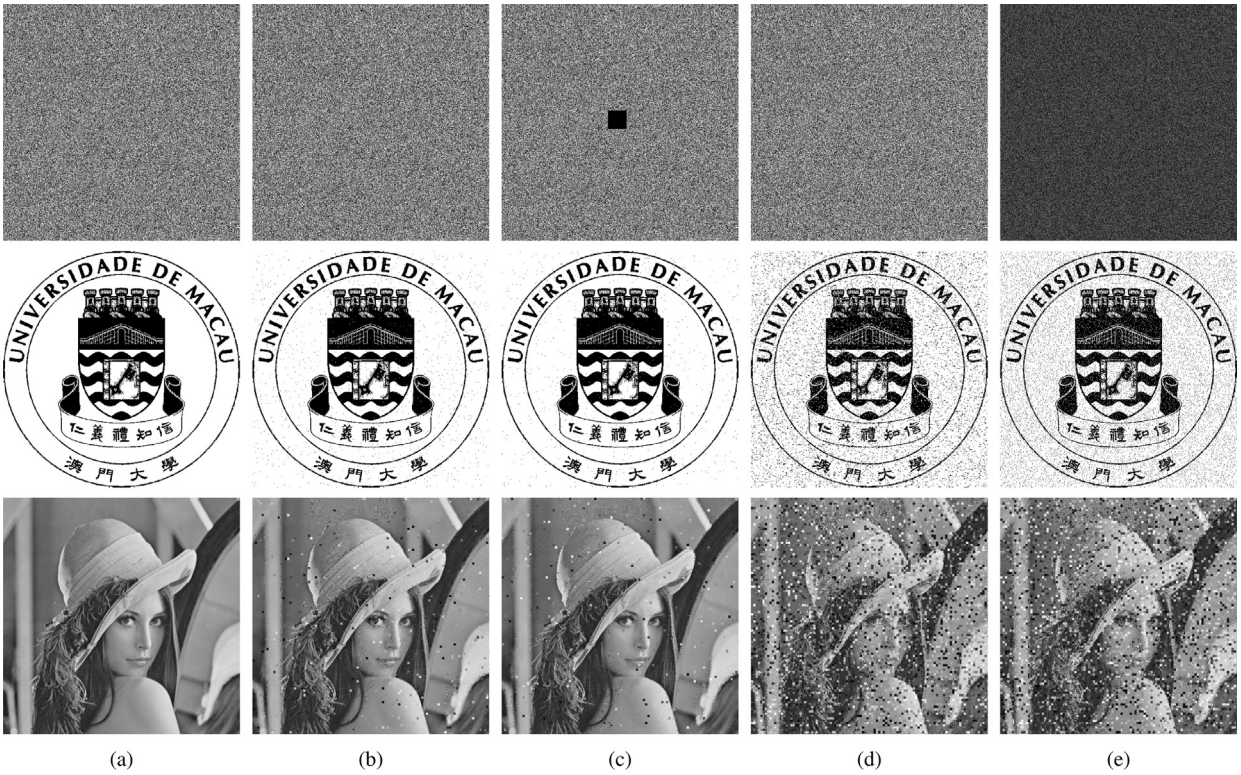

Fig. 13. Noise and data loss attacks. The first row shows the marked encrypted image (a) without noise; (b) with 1% Salt & Pepper noise; (c) with 40×40 image cut; (d) with the LSB plane removed; and (e) with the MSB plane removed, respectively. The second and third rows show the extracted secret data and recovered images, respectively.

observe, even one pixel difference between two original images yields totally different marked encrypted images. This can be verified by the image in Fig. 11(f).

To perform differential analysis in secret data, we embed two secret data into the same original image with the same security keys. Here, two secret data are the same except for one bit difference. The simulation results are shown in Fig. 12. As can be seen, even though the original images are the same, a tiny difference in secret data also results in totally different marked encrypted images.

Therefore, it is extremely difficult for the hackers to obtain any information from the correlations of changes between the plaintexts and ciphertexts. This is because even a tiny difference in the plaintexts (the original image or secret data) will result in a totally different marked encrypted image. This proves the effectiveness of BBE-RDHEI in against to differential attack.

6.2. Robustness analysis

In real applications, image may inevitably experience some noise and data loss during transmission. Thus, the RDHEI algorithm should have the ability of resisting noise and data loss attacks.

Fig. 13 shows the performance of BBE-RDHEI in withstanding the noise and data loss attacks. We choose the gray-scale Lena image and binary university logo to be the original image and secret data, respectively. In this experiment, we set the block size 5×5 and embedding rate r=1 bpp. Fig. 13 shows extracted secret data and recovered images from the marked encrypted images with various attacks of adding 1% Salt & Pepper noise, applying a 40×40 data cutting, removing the LSB and MSB planes, respectively. We can obverse that, BBE-RDHEI can successfully recover most of the original image and secret data. This is because BBE-RDHEI uses a bit-level scrambling to spread out the embedded secret data and important information of the original image to the entire marked encrypted image. However, other methods such as the ones in [21] and [14] suffer from serious secret data or original image content loss when a part of the LSB or MSB planes is lost during transmission.

7. Conclusion

In this paper, we have proposed a binary block embedding (BBE)

method for embedding messages in binary images. Based on BBE, we have proposed a reversible data hiding algorithm in encrypted images (BBE-RDHEI) in which BBE is utilized for reserving the bit space for embedding secret data. BBE-RDHEI employs a bit-level scrambling process after secret data embedding to spread embedded secret data to the entire marked encrypted image. A security key design mechanism is proposed to enhance its security level. Both BBE and BBE-RDHEI have been proved to be reversible. Simulations and comparisons have shown that BBE-RDHEI outperforms other existing methods in terms of the embedding rate and PSNR results of the decrypted images. Security analysis has demonstrated the robustness of BBE-RDHEI in against different attacks.

Acknowledgement

This work was supported in part by the Macau Science and Technology Development Fund under Grant FDCT/016/2015/A1 and by the Research Committee at University of Macau under Grants MYRG2014-00003-FST and MYRG2016-00123-FST.

References

[1] Z. Ni, Y.-Q. Shi, N. Ansari, W. Su, Reversible data hiding, IEEE Trans. Circuits Syst. Video Technol. 16 (3) (2006) 354–362.
[2] X. Li, B. Li, B. Yang, T. Zeng, General framework to histogram-shifting-based reversible data hiding, IEEE Signal Process. Lett. 22 (6) (2013) 2181–2191.
[3] P. Tsai, Y.-C. Hu, H.-L. Yeh, Reversible image hiding scheme using predictive coding and histogram shifting, Signal Process. 89 (6) (2009) 1129–1143.
[4] W. Hong, T.-S. Chen, C.-W. Shiu, Reversible data hiding for high quality images using modification of prediction errors, J. Syst. Softw. 82 (11) (2009) 1833–1842.
[5] J. Tian, Reversible data embedding using a difference expansion, IEEE Trans. Circuits Syst. Video Technol. 13 (8) (2003) 890–896.
[6] A. Alattar, Reversible watermark using the difference expansion of a generalized integer transform, IEEE Trans. Image Process. 13 (8) (2004) 1147–1156.
[7] H.-J. Kim, V. Sachnev, Y.Q. Shi, J. Nam, H.-G. Choo, A novel difference expansion transform for reversible data embedding, IEEE Trans. Inf. Forensics Secur. 3 (3) (2008) 456–465.
[8] D. Coltuc, J.-M. Chassery, Very fast watermarking by reversible contrast mapping, IEEE Signal Process. Lett. 14 (4) (2007) 255–258.
[9] X. Wang, X. Li, B. Yang, Z. Guo, Efficient generalized integer transform for reversible watermarking, IEEE Signal Process. Lett. 17 (6) (2010) 567–570.
[10] F. Peng, X. Li, B. Yang, Adaptive reversible data hiding scheme based on integer transform, Signal Process. 92 (1) (2012) 54–62.
[11] M.C.W. Puech, O. Strauss, A reversible data hiding method for encrypted images, Security, Forensics, Steganography, and Watermarking of Multimedia Contents X, in: Proceedings of SPIE 6819.
[12] X. Zhang, Reversible data hiding in encrypted image, IEEE Signal Process. Lett. 18 (4) (2011) 255–258.
[13] W. Hong, T.-S. Chen, H.-Y. Wu, An improved reversible data hiding in encrypted images using side match, IEEE Signal Process. Lett. 19 (4) (2012) 199–202.
[14] X. Wu, W. Sun, High-capacity reversible data hiding in encrypted images by prediction error, Signal Process. 104 (2014) 387–400.
[15] X. Zhang, Separable reversible data hiding in encrypted image, IEEE Trans. Inf. Forensics Secur. 7 (2) (2012) 826–832.
[16] X. Zhang, Z. Qian, G. Feng, Y. Ren, Efficient reversible data hiding in encrypted images, J. Vis. Commun. Image Represent. 25 (2) (2014) 322–328.
[17] Z. Qian, X. Han, X. Zhang, Separable reversible data hiding in encrypted images by n-nary histogram modification, in: Proceedings of the Third International Conference on Multimedia Technology, 2013, pp. 869–876.
[18] Z. Qian, X. Zhang, W. Shuozhong, Reversible data hiding in encrypted JPEG bitstream, IEEE Trans. Multimed. 16 (5) (2014) 1486–1491.
[19] Y.-C. Chen, C.-W. Shiu, G. Horng, Encrypted signal-based reversible data hiding with public key cryptosystem, J. Vis. Commun. Image Represent. 25 (5) (2014) 1164–1170.
[20] X. Zhang, J. Wang, Z. Wang, H. Cheng, Lossless and reversible data hiding in encrypted images with public key cryptography, IEEE Transactions on Circuits and Systems for Video Technology PP (99).
[21] K. Ma, W. Zhang, X. Zhao, N. Yu, F. Li, Reversible data hiding in encrypted images by reserving room before encryption, IEEE Trans. Inf. Forensics Secur. 8 (3) (2013) 553–562.
[22] L. Luo, Z. Chen, M. Chen, X. Zeng, Z. Xiong, Reversible image watermarking using interpolation technique, IEEE Trans. Inf. Forensics Secur. 5 (1) (2010) 187–193.
[23] W. Zhang, K. Ma, N. Yu, Reversibility improved data hiding in encrypted images, Signal Process. 94 (2014) 118–127.
[24] N. Otsu, A threshold selection method from gray-level histograms, IEEE Trans. Syst. Man Cybern. 9 (1) (1979) 62–66.
[25] Y. Zhou, L. Bao, C.P. Chen, A new 1D chaotic system for image encryption, Signal Process. 97 (2014) 172–182.
[26] Y. Zhou, W. Cao, C.P. Chen, Image encryption using binary bitplane, Signal Process. 100 (2014) 197–207.
[27] H.W. Wang, Shuozhong, Cyber warfare: Steganography vs. steganalysis, Communications of the ACM 47.
[28] W. Diffie, M.E. Hellman, New directions in cryptography, IEEE Trans. Inf. Theory 22 (6) (1976) 644–654.
[29] F.B.M. Barni, T. Furon, A general framework for robust watermarking security, Signal Process 83 (10) (2003) 2069–2084.
[30] F. Cayre, C. Fontaine, T. Furon, Watermarking security: theory and practice, IEEE Trans. Signal Process. 53 (10) (2005) 3976–3987.
[31] R. Anderson, Security engineering: A Guide to Building Dependable Distributed Systems, Wiley.