Bi-directional Block Encoding for Reversible Data Hiding over Encrypted Images

XINGYU LIU and ZHONGYUN HUA, Harbin Institute of Technology, Shenzhen, China
SHUANG YI, Southwest University of Political Science and Law, China
YUSHU ZHANG, Nanjing University of Aeronautics and Astronautics, China
YICONG ZHOU, University of Macau, China

Reversible data hiding over encrypted images (RDH-EI) technology is a viable solution for privacy-preserving cloud storage, as it enables the reversible embedding of additional data into images while maintaining image confidentiality. Since the data hiders, e.g., cloud servers, are willing to embed as much data as possible for storage, management, or other processing purposes, a large embedding capacity is desirable in an RDH-EI scheme. In this article, we introduce a novel bi-directional block encoding (BDBE) method, which, for the first time, encodes the distances of values in a binary sequence from both ends. This approach allows for encoding images with smaller sizes compared to traditional and state-of-the-art encoding methods. Leveraging the BDBE technique, we propose a high-capacity RDH-EI scheme. In this scheme, the content owner initially predicts the image pixels and then employs BDBE to encode the prediction errors, creating space for data embedding. The resulting encoded data are subsequently encrypted using a secure stream cipher, such as the Advanced Encryption Standard, before being transmitted to a data hider. The data hider can embed confidential information within the encrypted image for the purposes of storage, management, or other processing. Upon receiving the data, an authorized receiver can accurately recover the original image and the embedded data without any loss. Experimental results demonstrate that our RDH-EI scheme achieves a significantly larger embedding capacity compared to several state-of-the-art schemes.

CCS Concepts: • **Security and privacy → Digital rights management;**

Additional Key Words and Phrases: Image encoding, reversible data hiding, encrypted image

ACM Reference Format:
Xingyu Liu, Zhongyun Hua, Shuang Yi, Yushu Zhang, and Yicong Zhou. 2024. Bi-directional Block Encoding for Reversible Data Hiding over Encrypted Images. *ACM Trans. Multimedia Comput. Commun. Appl.* 20, 5, Article 149 (February 2024), 23 pages. https://doi.org/10.1145/3638771

This work was supported by the National Natural Science Foundation of China under Grants 62071142 and 62002301, the Guangdong Basic and Applied Basic Research Foundation under Grant 2021A1515011406, the Science and Technology Research Program of Chongqing Municipal Education Commission under Grants KJQN202200303 and KJQN201900310, the Research Foundation of Southwest University of Political Science and Law under Grant 2018XZQN-31, and the Foundation for Science and Technology Innovation of Shenzhen under Grant RCBS20210609103708014.
Authors' addresses: X. Liu and Z. Hua (Corresponding author), School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen Guangdong 518055, China; e-mails: {liuxyuh, huazyum}@gmail.com; S. Yi, Engineering Research Center of Forensic Science, Chongqing Education Committee, College of Criminal Investigation, Southwest University of Political Science and Law, Chongqing 401120, China; e-mail: yishuang@swupl.edu.cn; Y. Zhang, College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China; e-mail: yushu@nuaa.edu.cn; Y. Zhou, Department of Computer and Information Science, University of Macau, Macau 999078, China; e-mail: yicongzhou@um.edu.mo.

ACM 1551-6857/2024/02-ART149
https://doi.org/10.1145/3638771

1 INTRODUCTION

Nowadays, a huge number of images are produced from different imaging devices such as smart phones and digital cameras, and the growth is accelerated with the fast development of various Internet of Things applications [12]. The huge number of images cause image owners a heavy pressure for image storage, management or other processing. Cloud service is a natural choice for handling these explosively growth images [24, 26]. Data privacy is one of the most concerned points in image-based cloud services, since the image owners are unwilling to share some secret images, e.g., medical images, to the cloud servers or any unauthorized third-party. Then these images are usually encrypted before being uploaded to the cloud servers [19, 25]. However, the cloud servers would like to embed some additional data into the encrypted images for the purposes of storage, management or other processing. **Reversible data hiding over encrypted images (RDH-EI)** is an effective technique to accommodate this application, since it can embed some additional data into an encrypted image and completely recover the embedded data as well as the original image [6, 15, 27]. An RDH-EI scheme involves three participants: content owner, data hider, and receiver. The content owner encrypts an original image using an encryption key before transmitting the image to the data hider. The data hider embeds additional data into the encrypted image without accessing the content of the original image. The receiver can recover the original image and extract the embedded data using the encryption key and data hiding key, respectively. To protect image content and embed more data, an effective RDH-EI scheme is desired to have a large embedding capacity and a high security level [5, 9, 22].

Up to now, many RDH-EI schemes have been developed and they can be roughly classified into two categories: **vacating room after encryption (VRAE)** [14, 16, 35, 36] and **reserving room before encryption (RRBE)** [11, 13, 29, 34]. For the VRAE-based schemes, the embedding room is vacated by the data hider. The content owner directly encrypts the original image and then sends the encrypted image to the data hider, while the data hider vacates embedding room in the encrypted image for embedding data [2, 10, 31]. For example, in 2008, Puech et al. proposed the first VRAE-based scheme [14], in which the content owner encrypts the original image using the **advanced encryption standard (AES)**, while the data hider embeds additional data in each block of the encrypted image by bit substitution. The receiver can extract the embedded data and recover the image by analyzing the local standard deviation of image pixels. Several years later, Zhang et al. proposed another VRAE-based scheme [35], in which the image is encrypted using a stream cipher, and the additional data are embedded by flipping the low significant bits of some pixels in the image block. However, errors may occur in data extraction and image restoration. Qian et al. proposed a new scheme that can achieve a large embedding capacity [16], as much as 0.3 *bpp* for the image *Lena*. These schemes encrypt images using some existing stream ciphers, which can well protect the image content. However, the encrypted images by these stream ciphers have large information entropy scores and cannot be vacated too much embedding capacity [36]. To remain more pixel redundancy for a larger embedding capacity, some schemes use lightweight encryption strategies, e.g., bitwise XOR [2], block permutation [10], and co-modulation [31], to encrypt the images. However, these encryption strategies can only achieve visual security and their security strengths are very limited for protecting the image confidentiality [7, 18]. Besides, their embedding capacity cannot be very large, since some pixel correlations are eliminated during encrypting.

To achieve larger embedding capacity, Ma et al. proposed the first RRBE-based RDH-EI scheme [11], in which the embedding capacity is preserved by content owners in the plain image. Since the high pixel redundancy on the plain image is utilized, the RRBE-based RDH-EI schemes can obtain larger embedding capacity compared to the VRAE-based schemes. Several years later, some RRBE-based RDH-EI schemes using image coding and prediction errors have been developed [3, 17, 28, 32, 33]. For example, Yi et al. proposed a new binary block encoding method for

RDH-EI [30]. All bit planes of the original image are first divided into fixed-size blocks and then encoded using a binary block method for preserving embedding capacity. After being encrypted by a stream cipher, the encrypted image codes are sent to data hider for data hiding. Chen et al. proposed a new RDH-EI scheme in Reference [3], in which a pre-processing of calculating prediction errors is performed on the image to improve the proportion of zeros. By improving the block coding method in Reference [30], the scheme can effectively encode the prediction errors and thus achieves a large embedding capacity, as much as 2.59 *bpp* for image *Lena*. Xu et al. [28] performed pixel prediction using the MED predictor and encoded each bit plane of the prediction errors using the method in Reference [3], in which an adaptive strategy is designed for block division to improve the coding effect. Yin et al. [32] first calculated the prediction errors of the image and then preserved embedding capacity using extended run-length code [4]. Qiu et al. [17] employed the MED predictor to predict image and classified the pixels into $2T + 1$ groups using a threshold T. Then the pixel category sequence of the image is compressed using arithmetic coding. However, determining the appropriate threshold T requires an exhaustive enumeration of values from 1 to 255, resulting in high time complexity. For these RRBE-based RDH-EI schemes, the high pixel redundancy of plain image is used to preserve embedding capacity and any stream cipher can be used to encrypt the images. They can achieve both large embedding capacity and high security level. However, many existing RRBE-based RDH-EI schemes have very complex embedding capacity preserving methods [20, 32], which incur heavy computation costs to the content owners, especially for the performance-limited terminal devices such as some Internet of Things devices. Besides, some schemes do not have very large embedding capacity [3, 30].

In this article, we propose the **bi-directional block encoding (BDBE)** method. Unlike existing distance encoding methods, BDBE can encode a binary sequence from both ends, resulting in higher encoding efficiency compared to classical and recent block encoding methods. We present a novel **BDBE-based RDH-EI scheme (BDBE-RDHEI)**. This scheme allows the content owner to preprocess an image using an improved mixed predictor to calculate prediction errors. The prediction errors are then encoded to preserve embedding capacity. After being encrypted using the AES, the encrypted image codes are transmitted to a data hider for data embedding. An authorized receiver can extract the embedded data and recover the original image using the data hiding key and encryption key, respectively. Experimental results demonstrate that our scheme achieves a larger embedding capacity than several state-of-the-art methods. The main contributions of this article are summarized as follows:

- We propose BDBE as a novel block encoding method, which is the first work to encode the distances of values in a binary block from both ends. Due to the shorter distances when encoding from both sides, BDBE achieves superior encoding efficiency compared to classical and recent block encoding methods.
- We introduce a high-capacity RDH-EI scheme utilizing BDBE, which ensures separate and lossless data extraction and image recovery processes.
- We conduct a comprehensive evaluation, and the results confirm that our scheme guarantees image content confidentiality, with an embedding rate of 3.533 *bpp* for the image *Lena*. Comparative analysis showcases its larger embedding capacity compared to state-of-the-art methods in References [3, 13, 17, 28, 32, 33].

The rest of this article is organized as follows. Section 2 introduces the BDBE in detail. Section 3 describes the BDBE-RDHEI scheme. Section 4 compares BDBE with classical and latest encoding methods, provides a comprehensive evaluation of the BDBE-RDHEI scheme, and compares it with the state-of-the-art schemes. Section 5 concludes this article. The important notations used in this article are listed in Table 1.

Table 1. Key Notations

Notation	Description
S	The block size of a binary block
n_0, n_1	The numbers of 0s and 1s in a binary block
z	The minimum value of n_0 and n_1
ξ	The sign of the smaller proportion in a Type-II binary block
N_a	The threshold of a binary block with size $S \times S$
Str	The one-dimensional string format of a block
BN	The Huffman code of z
BD	The location information of the ξs in **Str**
$\mathbf{BD}_i$	The location information of the ith sign ξ
L_i	The length of $\mathbf{BD}_i$
BC	The code of a binary block
$e(i, j)$	The prediction error of the pixel $p(i, j)$
LM	A location map to identify whether a pixel satisfies $e(i, j) \in [-128, 127]$
T	A binary sequence to store the original values of the pixels whose $e(i, j) \notin [-128, 127]$
EI	A 9-bit error image that is divided into 18 bit-planes for compression
P_1	The proportion of 1s in a bit-plane of **EI**
k	The number of un-encodable bit-planes in **EI**
U	The k un-encodable bit-planes
C	The encoding results of $18 - k$ encodable bit-planes
SI	The side information
MI	The main information that consists of **T**, **C** and **U**
EMI	The encrypted results of **MI**
E	The encrypted image
EM	The marked encrypted image
$\|\|$	Bits connection

Table 2. Block Types and Codes

Condition	Type	Description	Block Code **BC**	Sign ξ
$z > N_a$	I	cannot embed data	1$\|\|$**Str**	—
$z = n_0 \le N_a$	II	most pixels are 1	0$\|\|$**BN**$\|\|$**BD**	0
$z = n_1 \le N_a$		most pixels are 0		1

2 BI-DIRECTIONAL BLOCK ENCODING

Recently, many binary sequence encoding methods were developed by encoding the distances of values [3, 30, 32]. However, all these methods encode distances from only one side and thus cannot use the least bits to present the distances. Here, we develop the BDBE method, which encodes the distances from both sides using the least bits.

2.1 BDBE

Suppose that a binary block is of size $S \times S$ and n_0 and n_1 are the numbers of zeros and ones within the block, respectively. Set $z = min\{n_0, n_1\}$ as the minimum value of n_0 and n_1. According to a threshold N_a, we can classify the block as an un-encodable block or an encodable block, which are called Type-I and Type-II, respectively, shown in Table 2. The symbol $||$ represents bits connection.

For a block, we convert it to a one-dimensional string **Str** by scanning the block from left to right and top to bottom. For a Type-I block with $z > N_a$, we use one bit 1 and **Str** to present it. For a Type-II block, we first use one bit sign ξ to present the sign of the smaller proportion within the

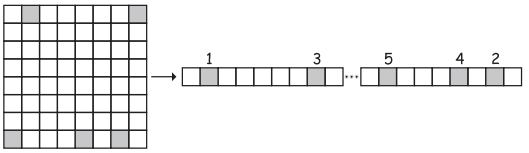

Fig. 1. Example of encoding order for a Type-II block.

block, i.e., $\xi = 0$ if $z = n_0$, and 1 otherwise. We then encode the parameter z to a fixed Huffman code **BN** with N_a-bits at most. When $z = 0$, set **BN** as '0'. When $z = N_a$, **BN** is set as N_a-bits '1's. When $0 < z < N_a$, **BN** is set as z-bits '1's and one bit '0'. For example, when $N_a = 3$, the Huffman codes **BN** of the numbers 0, 1, 2, and 3 are '0', '10', '110', and '111', respectively. Notice that there is no need to store an additional Huffman table for the parameter z.

When $z \neq 0$, we should encode the location information of the ξs in **Str**. We encode the location of each ξ bi-directionally, as shown in Figure 1. In the block on the left, the small gray squares represent the ξs to be encoded. Convert the block to a one-dimensional string **Str** by scanning the block from left to right and top to bottom and then count the ξs bi-directionally as indicated by the numbers. The detailed processes for generating the codes **BD** of a Type-II block can be described as follows:

— *Step 1:* For the ith sign ξ, if i is odd, then the encoding begins from the left end. Get the length of **Str** as *Len*. Suppose that the index of the leftmost sign ξ from left to right is I_i. Since $I_i \in [1, Len - (z - i)]$, encode I_i using $\max\{1, \lceil \log_2(Len - (z - i)) \rceil\}$ bits to obtain the code $\mathbf{BD}_i$. Update the **Str** by removing its first to I_ith elements counting from the left end.
— *Step 2:* For the ith sign ξ, if i is even, then the encoding begins from the right end. Get the length of **Str** as *Len*. Suppose that the index of the rightmost sign ξ from right to left is I_i. Then $I_i \in [1, Len - (z - i)]$ and use $\max\{1, \lceil \log_2(Len - (z - i)) \rceil\}$ bits to encode I_i to obtain $\mathbf{BD}_i$. Update the **Str** by removing its first to I_ith elements counting from the right end.
— *Step 3:* Iterate the *Step 1* and *Step 2* for all the ξs.
— *Step 4:* Concatenate the codes $\mathbf{BD}_1, \mathbf{BD}_2, \ldots, \mathbf{BD}_z$ to obtain the **BD** of the block.

Note that the **BD** is obtained by recording the locations of ξs bi-directionally, which is the core step of the BDBE. Finally, for a Type-II block, we can present it using one bit sign ξ and code **BC**, which includes the marker of encodable block '0', the Huffman code **BN**, and the distance string **BD**.

Algorithm 1 presents the pseudo-code of the BDBE for a block. To better show the processing details, we provide an illustrative example for five binary blocks with size 4×4 in Figure 2. Suppose that the threshold N_a for 4×4 block is 3. For the block in Figure 2(a), $z = n_1 = 4 > N_a$, thus it is a Type-I block, and the length of code **BC** is 17. For the blocks in Figure 2(b) and (c), the pixels are all ones or zeros, thus they are Type-II blocks with $z = 0$. Their codes **BC** are both '00', and they have one bit sign ξ, 0, and 1, respectively. For the block in Figure 2(d), it is a Type-II block with $z = 3, \xi = 1$. Encode the parameter z into **BN** = '111'. The location information of the three 1s, **BD**, includes $\mathbf{BD}_1$, $\mathbf{BD}_2$, and $\mathbf{BD}_3$, which are produced as follows:

— For $i = 1$, the encoding begins from the left end. The length of the **Str** is $Len = 16$. Considering that the rest two 1s are at the right side, then the maximum index of the current 1 is $Len-(z-i) = 16-(3-1) = 14$. Since the index $I_1 = 11$, we use $L_1 = \max\{1, \lceil \log_2 14 \rceil\} = 4$ bits to encode the distance $I_1 - 1 = 10$ and obtain $\mathbf{BD}_1$ = '1010'. Update the **Str** as **Str** = '10100'.
— For $i = 2$, the encoding begins from the right end. The length of the block string $Len = 5$. The maximum index $Len - (z - i) = 5 - (3 - 2) = 4$. Since the index $I_2 = 3$, use $L_2 = \max\{1, \lceil \log_2 4 \rceil\} = 2$ bits to encode its distance $I_2 - 1 = 2$ and obtain $\mathbf{BD}_2$ = '10'. Update the **Str** as **Str** = '10'.

ALGORITHM 1: The pseudo-code of the BDBE for a block.

Input: An $S \times S$ block, the threshold N_a.

Initialize block code **BC**, sign ξ.
Convert the block into a string **Str**.
Calculate n_0, n_1. $z \leftarrow \min\{n_0, n_1\}$.
if $z > N_a$ **then**
  $\xi \leftarrow$ null.
  BC $\leftarrow 1 \| \mathbf{Str}$.
else
  Calculate sign ξ.
  Encode z into **BN** using the Huffman table.
  BC $\leftarrow 0 \| \mathbf{BN}$.
  for $i \leftarrow 1$ to z **do**
    $Len \leftarrow$ Length(**Str**). // Retrieve the length.
    if $(i \mod 2) = 1$ **then**
      Find the index of the first ξ in the **Str** as I_i from the left end.
      Str $\leftarrow$ **Str**$[I_i + 1 : Len]$.
    else
      Find the index of the first ξ in the **Str** as I_i from the right end.
      Str $\leftarrow$ **Str**$[1 : Len - I_i]$.
    end if
    $L_i \leftarrow \max\{1, \lceil \log_2(Len - (z - i)) \rceil\}$.
    $\mathbf{BD}_i \leftarrow \text{dec2bin}(I_i - 1, L_i)$. // Transform decimal to binary.
    BC $\leftarrow$ **BC**$\|\mathbf{BD}_i$.
  end for
end if

Output: The block code **BC**, the sign ξ.

— For $i = 3$, the encoding begins from the left end. Then $Len = 2$. The maximum index $Len - (z - i) = 2 - (3 - 3) = 2$. The index $I_3 = 1$, and we then use $L_3 = \max\{1, \lceil \log_2 2 \rceil\} = 1$ bit to encode its distance $I_3 - 1 = 0$ and obtain $\mathbf{BD}_3 =$ '0'. Since there is no more sign $\xi = 1$ in **Str**, the encoding finishes.

Thus, for the block in Figure 2(d), its code **BC** = '01111010100' and sign $\xi = 1$. Using the same way, we can encode the block in Figure 2(e) as **BC** = '01101000010' and sign $\xi = 0$.

2.2 Discussion of the Threshold N_a

The performance of BDBE is sensitive to the chosen threshold N_a. An inappropriate value of N_a can lead to insufficient compression or excessive encoding of some blocks, ultimately degrading the overall BDBE encoding performance. In our scheme, the threshold N_a is to determine whether a binary block is Type-I or Type-II, as shown in Table 2. n_0 and n_1 present the numbers of 0s and 1s in a binary block, z indicates the minimum value of n_0 and n_1, and ξ is the sign of the smaller proportion within a Type-II block. Specifically, $\xi = 0$ if the number of 0s is smaller than the number of 1s; otherwise, $\xi = 1$. Type-I block indicates that the code length of the block is larger than the block size, thus the block cannot embed data, while Type-II block means that the code length of the block is smaller than the block size and thus the block can embed data.

For a Type-II block with block size $S \times S$, we use x to represent the number of ξs within the block. Then two bits are employed to indicate whether the block is Type-II and the value of ξ, which can be 0 or 1. x bits are used to encode the value x using Huffman code and a total number of $\sum_{i=1}^{x} L_i$ bits are used to encode the positions of all the ξs. For a Type-I block with block size $S \times S$, one bit

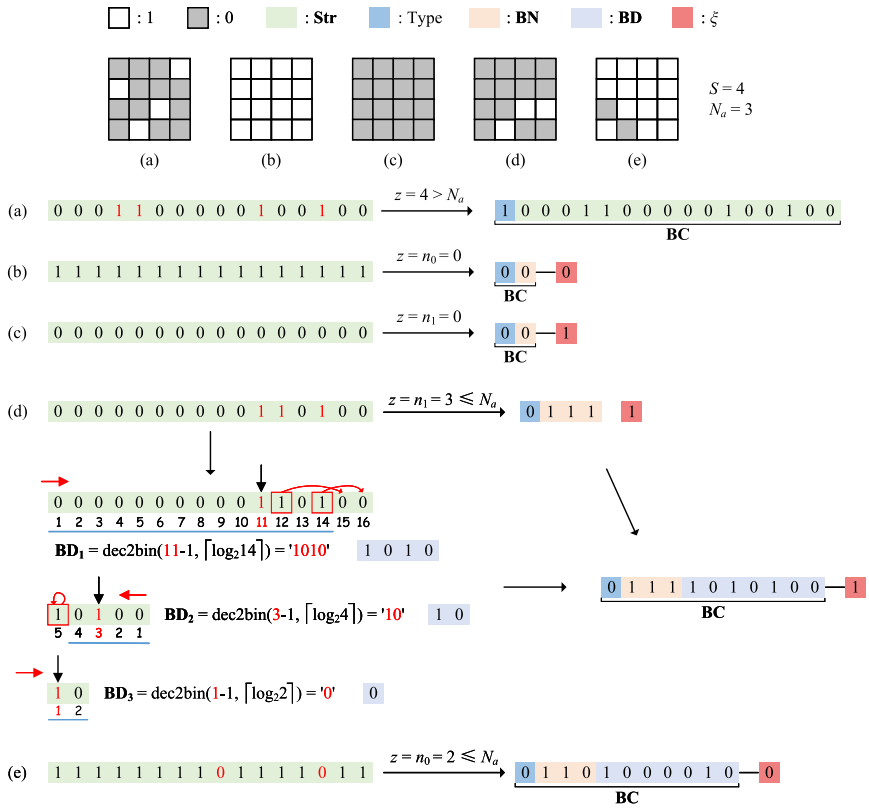

Fig. 2. BDBE examples with block size $S = 4$ and threshold $N_a = 3$. (a) Type-I block; ((b) and (c)) Type-II blocks with all ones and zeros, respectively; (d) Type-II block with $z = n_1 = 3$; (e) Type-II block with $z = n_0 = 2$.

is employed to indicate that the block is Type-I and S^2 bits are used to store the binary block as it is not encoded.

It is important to note that the length of the Type-II block's code should not exceed that of a Type-I block. Then we can get the equation

$$2 + x + \sum_{i=1}^{x} L_i \leq 1 + S^2. \tag{1}$$

Since the threshold N_a is to determine whether a binary block is Type-I or Type-II, it can be calculated as

$$N_a = \arg\max_{x} \left\{ S^2 - \left(x + \sum_{i=1}^{x} L_i \right) > 0 \right\}. \tag{2}$$

Suppose that all these ξs are uniformly distributed in the block. Then, the bit number required to present the ith sign ξ can be calculated as

$$L_i = \begin{cases} \lceil \log_2(S^2 - (x-1)) \rceil, & i = 1; \\ \lceil \log_2(S^2 - \lfloor \frac{S^2}{2x} \rfloor - (x-2)) \rceil, & i = 2; \\ \lceil \log_2(S^2 - \lfloor \frac{(i-2)S^2}{x} \rfloor - (x-i)) \rceil, & 3 \leq i \leq x. \end{cases} \tag{3}$$

Table 3. The Relationship between the Block Size $S \times S$ and Threshold N_a

S	N_a	S	N_a	S	N_a
4	3	32	102	256	4112
8	9	64	342	512	13604
16	31	128	1175	...	...

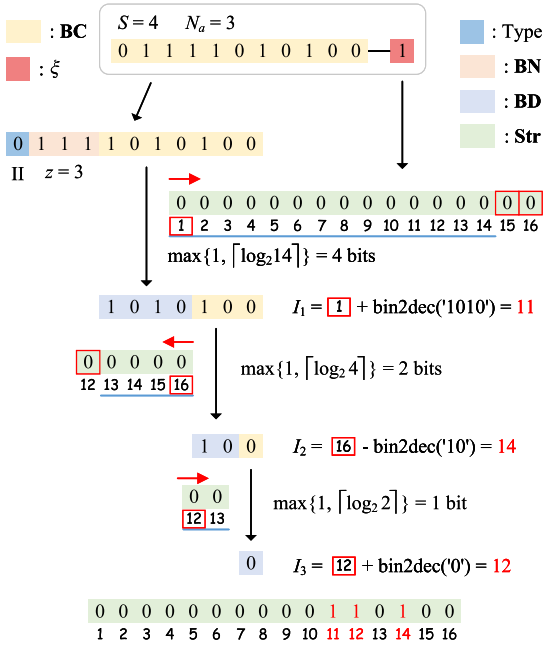

Fig. 3. An illustrative example of the inverse BDBE.

From Equation (2) and Equation (3), we can get the relationship between the threshold N_a and block size S, which is listed in Table 3.

2.3 Recovery of BDBE

Using the code **BC** and sign ξ, we can recover the original binary stream. From Table 2, the binary blocks with size $S \times S$ are divided into Type-I and Type-II. A block is a Type-I block if the first bit of **BC** is '1', while the block is a Type-II block if the bit is '0'. For a Type-I block, the following $S \times S$ bits in **BC** are the bits of the block. For a Type-II block, if the second bit of **BC** is '0', then we know $z = 0$, then easily recover it with sign ξ; otherwise, we divide the following **BC** into **BN** and **BD** to recover the Type-II block. The recovery processes are the opposite operations of the encoding processes and Algorithm 2 presents the pseudo-code of the recovery processes.

To better show the recover processes, we also give a numeral example of recovering the bit-stream in Figure 2(d) and Figure 3 shows the processed in detail. As can be seen, we can determine it is a Type-II block and $z = 3$, according to the code **BC** and N_a. Because $\xi = 1$, we know that the **BC** includes the indexes of three 1s, which can be recovered as follows:

— For $i = 1$, $r_l = 1, r_h = 16$, thus $Len = 16$ and $L_1 = \max\{1, \lceil \log_2(Len - (z - i)) \rceil\} = \max\{1, \lceil \log_2 14 \rceil\} = 4$ bits. From **BC**, obtain $\mathbf{BD}_1$ = '1010'$\rightarrow (10)_{10}$, then $I_1 = r_l + 10 = 11$. Update $r_l = 12$.

— For $i = 2$, $r_l = 12, r_h = 16$, thus $Len = 5$ and $L_2 = \max\{1, \lceil \log_2(Len - (z - i))\rceil\} = \max\{1, \lceil \log_2 4\rceil\} = 2$ bits. From **BC**, obtain $\mathbf{BD}_2 =$ '10'$\rightarrow (2)_{10}$, then $I_2 = r_h - 2 = 14$. Update $r_h = 13$.
— For $i = 3$, $r_l = 12, r_h = 13$, thus $Len = 2$ and $L_3 = \max\{1, \lceil \log_2(Len - (z - i))\rceil\} = \max\{1, \lceil \log_2 2\rceil\} = 1$ bit. From **BC**, obtain $\mathbf{BD}_3 =$ '0'$\rightarrow (0)_{10}$, then $I_3 = r_l + 0 = 12$. Update $r_l = 13$.

After setting the values at the positions of I_1, I_2, and I_3 as the sign ξ, we can recover the string **Str** of the original binary block.

ALGORITHM 2: The inverse BDBE for a block.

Input: Block size S, threshold N_a, code **BC** and sign ξ.
Initialize a string **Str** with length $S \times S$.
if BC[1] = 1 **then**
  Str $\leftarrow$ **BC**$[2 : S \times S + 1]$.
else
  BC $\leftarrow$ **BC**[2 : end].
  Str $\leftarrow$ Set all elements to $1 - \xi$.
  if BC[1] = 1 **then**
    $z \leftarrow$ the number of consecutive 1s in **BC**.
    $z \leftarrow \min\{z, N_a\}$.
    BC $\leftarrow$ **BC**$[\min\{z + 1, N_a\} + 1 :$ end].
    $r_l \leftarrow 1$. $r_h \leftarrow S \times S$.
    for $i \leftarrow 1$ to z **do**
      $Len \leftarrow r_h - r_l + 1$.
      $L_i \leftarrow \max\{1, \lceil \log_2(Len - (z - i))\rceil\}$.
      $\mathbf{BD}_i \leftarrow$ **BC**$[1 : L_i]$.
      BC $\leftarrow$ **BC**$[L_i + 1 :$ end].
      $I_i \leftarrow$ bin2dec($\mathbf{BD}_i$). // Transform binary to decimal.
      if $(i \mod 2) = 1$ **then**
        $I_i \leftarrow r_l + I_i$.
        $r_l \leftarrow I_i + 1$.
      else
        $I_i \leftarrow r_h - I_i$.
        $r_h \leftarrow I_i - 1$.
      end if
      set the I_ith element in **Str** as ξ.
    end for
  end if
end if
Transform **Str** into a block with size $S \times S$.

Output: The block with size $S \times S$.

3 BDBE-BASED RDH-EI

In this section, we propose BDBE-RDHEI, whose flow diagram is shown in Figure 4. For a content owner, he/she first uses an improved mixed predictor to the original image to generate a 9-bit plane image **EI**. Using the BDBE, the **EI** is encoded into two bit streams, **SI** and **MI**. The content owner then encrypts the bitstream **MI** to obtain an encrypted image **E**. A data hider can embed additional secret data into the encrypted image to get a marked encrypted image **EM**. An authorized receiver can extract the embedded data using the data hiding key, and recover the original image using the encryption key.

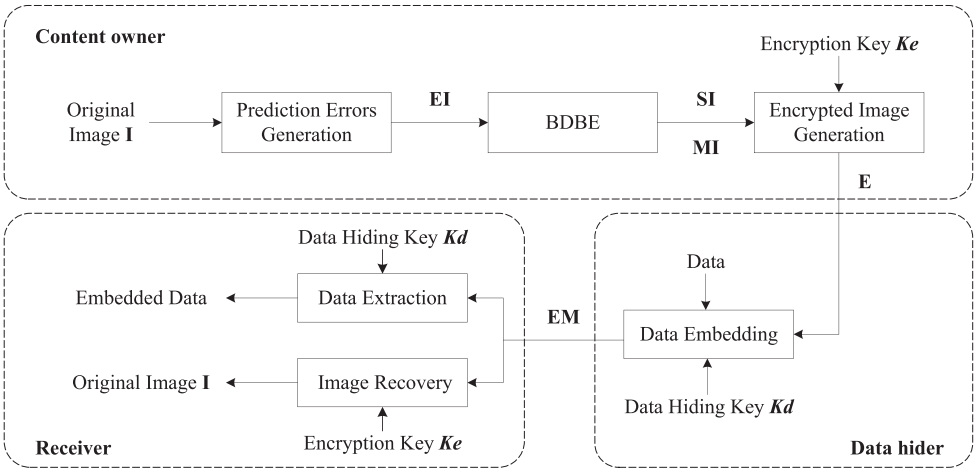

Fig. 4. Structure of the proposed BDBE-RDHEI.

3.1 Prediction Errors Generation

An improved predictor, derived from the MED predictor and a modified version of the GAP predictor [23], is employed to process the image. This predictor efficiently combines the strengths of both predictors. For an 8-bit image **I** with size $M \times N$, the predicted value of pixel $p(i, j)$ is calculated based on its neighboring pixels, as shown in Figure 5. Based on the pixel locations, the pixels in the entire image are divided into three cases:

— *Case 1*: Pixels in the first row, first column, and last column. The prediction value is calculated as

$$\hat{p}(i,j) = \begin{cases} 0, & \text{for } i = 1, j = 1; \\ p_1, & \text{for } i = 1, j \neq 1; \\ p_4, & \text{for } i \neq 1, j = 1 \text{ or } j = N. \end{cases} \tag{4}$$

— *Case 2*: Pixels in the second row and second column, excluding those in *Case 1*. Using the MED predictor, the prediction value is calculated as

$$\hat{p}(i,j) = \begin{cases} \max(p_1, p_4), & p_3 \leq \min(p_1, p_4); \\ \min(p_1, p_4), & p_3 \geq \max(p_1, p_4); \\ p_1 + p_4 - p_3, & \text{otherwise.} \end{cases} \tag{5}$$

— *Case 3*: Pixels in the rest regions. Using the modified GAP predictor, the prediction value is calculated as

$$\hat{p}(i,j) = \begin{cases} p_1, & \Delta > 80; \\ (p_1 + u)/2, & 32 < \Delta \leq 80; \\ (p_1 + 3 * u)/4, & 8 < \Delta \leq 32; \\ u, & -8 \leq \Delta \leq 8; \\ (p_4 + 3 * u)/4, & -32 \leq \Delta < -8; \\ (p_4 + u)/2, & -80 \leq \Delta < -32; \\ p_4, & \text{otherwise,} \end{cases} \tag{6}$$

where $\Delta = (|p_1 - p_3| + |p_3 - p_6| + |p_4 - p_7| + |p_5 - p_8|) - (|p_0 - p_1| + |p_2 - p_3| + |p_3 - p_4| + |p_4 - p_5|)$, and $u = (p_1 + p_4)/2 + (p_5 - p_3)/4$. Note that the prediction result is a decimal and we use ceiling rounding to convert it into an integer.

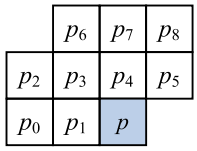

Fig. 5. A neighborhood of the target pixel p, i.e., $p(i, j)$.

Table 4. Block Types and Codes in the BDBE-RDHEI

Condition	Type	Description	Code **BC**
$n_1 > N_a$	I	cannot embed data	1‖**Str**
$0 \le n_1 \le N_a$	II	most pixels are 0	0‖**BN**‖**BD**

The prediction error of $p(i, j)$ can be calculated as

$$e(i, j) = p(i, j) - \hat{p}(i, j). \tag{7}$$

According to whether a prediction error is within the range $[-128, 127]$ or not, all the prediction errors are divided into two classes. A location map **LM** is used to identity the class of the prediction errors. When $e(i, j) \in [-128, 127]$, $\mathbf{LM}(i, j) = 0$; otherwise, $\mathbf{LM}(i, j) = 1$. For a prediction error $e(i, j) \in [-128, 127]$, 8 bits can be used to present its value by

$$e(i, j) = \begin{cases} 2 \times e(i, j), & e(i, j) \ge 0; \\ -2 \times e(i, j) - 1, & e(i, j) < 0. \end{cases} \tag{8}$$

When a prediction error $e(i, j) \notin [-128, 127]$, we first store the original pixel $p(i, j)$ to a binary sequence **T** and then set the prediction error as 0. After being processed, all the prediction errors can be represented by 8 bits. Combined with the location map, a 9-bit error image **EI** is generated.

3.2 Encoding by BDBE

For the error image **EI**, the proportion of 0s in a bit plane is much larger than 1s. Thus, the situation $z = n_0$ in Table 2 is less likely to happen and we can remove this situation. Then the sign bit ξ is unnecessary and we can change Table 2 as Table 4.

In our scheme, each bit plane of **EI** is divided into two bit-planes: the first $M/2$ rows and the last $M/2$ rows. As a result, the 9-bit error image **EI** consists of 18 bit-planes. These bit-planes are sequentially labeled in the order of the 9th bit plane's first $M/2$ rows, 9th bit plane's last $M/2$ rows, 8th bit plane's first $M/2$ rows, $\cdots$, and the 1st bit plane's last $M/2$ rows, with labels ranging from 1 to 18. The BDBE's performance varies with different bit-plane block sizes. Typically, larger block sizes are preferred for bit-planes with fewer 1s to achieve superior results. To determine the optimal block size, we conducted experiments using the "BOWS-2" database. For each image in the database, we calculate its error image **EI** and obtain 18 bit-planes. Then we compress all images' bit-planes using block sizes of 4, 8, 16, 32, 64, 128, and 256, respectively. We use N_1 to denote the number of 1s in a bit-plane. Then, for bit-planes with the same N_1, we separately count the numbers of bit-planes whose optimal encoding block size is 4, 8, 16, 32, 64, 128, and 256. If the bit-plane with block size S has the largest number, then we consider S as the optimal block size for encoding bit-planes with N_1. As N_1 increases, the optimal block size S gradually decreases. We choose the critical N_1 values for the change of S and obtain the correlation of N_1 and S. To apply the block size selection method across bit-planes of different sizes, we utilize the proportion of 1s (P_1) in a bit-plane as the classification criterion, rather than the exact number of 1s (N_1). Finally,

we obtain the correlation between the optimal block size S and P_1 as

$$S = \begin{cases} 256, & \text{for } P_1 \in [0, 6.4850 \times 10^{-5}]; \\ 128, & \text{for } P_1 \in (6.4850 \times 10^{-5}, 2.4796 \times 10^{-4}]; \\ 64, & \text{for } P_1 \in (2.4796 \times 10^{-4}, 9.2697 \times 10^{-4}]; \\ 32, & \text{for } P_1 \in (9.2697 \times 10^{-4}, 3.4599 \times 10^{-3}]; \\ 16, & \text{for } P_1 \in (3.4599 \times 10^{-3}, 1.4309 \times 10^{-2}]; \\ 8, & \text{for } P_1 \in (1.4309 \times 10^{-2}, 8.6376 \times 10^{-2}]; \\ 4, & \text{for } P_1 \in (8.6376 \times 10^{-2}, 3.4893 \times 10^{-1}]; \\ 1, & \text{for } P_1 \in (3.4893 \times 10^{-1}, 1], \end{cases} \tag{9}$$

in which the block size $S = 1$ means the bit-plane is not encoded. Note that the bit-plane is not encoded if the length of the encoded result is larger than total bits of original bit-plane.

Finally, we can encode each bit-plane of **EI** by BDBE using the following steps:

— *Step 1*: Calculate the P_1;
— *Step 2*: Get the block size S from Equation (9);
— *Step 3*: Divide the bit-plane into blocks with size $S \times S$;
— *Step 4*: Encode each block according to Table 4;
— *Step 5*: Combine the codes of all the image blocks to a encoded result of the bit-plane.

It is obvious that when the proportion of 1s within a bit-plane P_1 is large than 3.4893×10^{-1}, the bit-plane is not encoded. Thus, the 18 bit-planes of the error image **EI** can be divided into encodable bit-planes and un-encodable bit-planes. Suppose that k bit-planes of the error image are un-encodable and the rest $18 - k$ bit-planes are encodable. For the k un-encodable bit-planes, k bitstreams $\mathbf{U} = \{\mathbf{U}_{i_1}, \mathbf{U}_{i_2}, \ldots, \mathbf{U}_{i_k}\}$ are generated using the raster order of the k un-encodable bit-planes, and their identities are $i_1, i_2, \cdots, i_k$. For the $18 - k$ encodable bit-planes, the BDBE encoding results are $\mathbf{C} = \{\mathbf{C}_{j_1}, \mathbf{C}_{j_2}, \ldots, \mathbf{C}_{j_{18-k}}\}$ and their identities are $j_1, j_2, \ldots, j_{18-k}$. To recover the original image, the following information are required.

— 5 bits to present the value of the k;
— $5k$ bits to present the identities of the k un-encodable bit-planes $i_1, i_2, \cdots, i_k$;
— $3(18 - k)$ bits to present the block sizes of the $18 - k$ encodable bit-planes. Because there are 7 types of block sizes and each block size is presented using 3 bits;
— $(18 - k)\lceil \log_2(MN/2) \rceil$ bits to present the binary code lengths of $\mathbf{C} = \{\mathbf{C}_{j_1}, \mathbf{C}_{j_2}, \ldots, \mathbf{C}_{j_{18-k}}\}$. Since the maximum length of the binary code for each bit-plane is $M/2 \times N$, its length can be presented using $\lceil \log_2(MN/2) \rceil$ bits;
— $\lceil \log_2(MN) \rceil$ bits to present the number of pixels, whose prediction error $e(i, j) \notin [-128, 127]$;
— The binary sequence **T** that contains the original pixels whose prediction errors are not within $[-128, 127]$;
— The BDBE results **C** of the $18 - k$ encodable bit-planes;
— The k bitstreams **U** of the k un-encodable bit-planes.

These required information are divided into two parts. The first five items form a binary sequence, namely side information **SI**. The last three items construct the main information **MI** = **T**||**C**||**U**.

3.3 Encrypted Image Generation

After encoding the original image into two binary sequences, **SI** and **MI**, one can encrypt the **MI** using any existing encryption standard, e.g., the AES, to obtain the encrypted bit stream **EMI**. Then, the encrypted image is generated by arranging the bitstreams **SI** and **EMI**. Finally, the binary

sequences **SI** and **EMI** are placed at the 8 bit planes of image from the most significant bit plane to the least significant bit plane, and the room that is not occupied is vacated for data embedding.

3.4 Data Embedding

When a data hider receives an encrypted image **E**, he/she can embed some additional data into the vacated room. First, according to the description in Section 3.2, he/she can extract the **SI** from the most significant bit plane of the encrypted image **E**, and then find the position of the vacated room. After finding the vacation room, he/she can embed the additional data. Note that the additional data can also be encrypted using an existing encryption standard with a data hiding key before being embedded into the encrypted image. After data embedding, a marked encrypted image **EM** is generated.

3.5 Data Extraction and Image Recovery

A receiver can separately extract the embedded data and recover the original image using the related keys.

3.5.1 Data Extraction with Data Hiding Key. For a marked encrypted image **EM**, if the receiver has the corresponding data hiding key, then he/she can correctly extract these additional data. According to the image encoding strategy, the side information **SI** is first extracted and the beginning position of the embedded data can be obtained. Then the embedded data can be extracted. Using the data hiding key, one can recover the additional data.

3.5.2 Image Recovery with Encryption Key. From a marked encrypted image **EM**, one can first obtain the side information **SI** and encrypted main information **EMI** of the original image. Then the image recovery can be described as follows:

- *Step 1*: Recover the main information **MI** from **EMI** using the encryption key.
- *Step 2*: The structure of the **SI** is described in Section 3.2, from which we can get the value of k, the identities of the k un-encodable bit-planes and $18-k$ encodable bit-planes, the used block sizes for encoding the $18-k$ bit-planes, the lengths of binary codes of the $18-k$ encodable bit-planes $\mathbf{C} = \{\mathbf{C}_{j_1}, \mathbf{C}_{j_2}, \ldots, \mathbf{C}_{j_{18-k}}\}$, and the number of pixels whose prediction errors are not within $[-128, 127]$. Note that the number is 1/8 of the length of binary sequence **T**.
- *Step 3*: According to the lengths of **T** and **C**, divide **MI** into three items: **T**, **C** and **U**.
- *Step 4*: Using the block sizes for encoding the $18-k$ bit-planes, perform inverse BDBE and obtain the original bit-planes of the $18-k$ encodable bit-planes. Combined with the k bit-streams **U**, the 9-bit error image **EI** is recovered.
- *Step 5*: Recover the original image **I**. According to the 9th bit plane, namely the location map **LM**, one can directly recover the pixels whose prediction errors $e(i,j) \notin [-128, 127]$ from the binary sequence **T**. For these pixels with $e(i,j) \in [-128, 127]$, their prediction errors are the 8 least significant bits of **EI**. First, recover the original prediction errors using the inverse operation of Equation (8), which is expressed as

$$e(i,j) = \begin{cases} e(i,j)/2, & (e(i,j) \mod 2) = 0; \\ -(e(i,j)+1)/2, & (e(i,j) \mod 2) \neq 0. \end{cases} \tag{10}$$

  Then, all pixels are recovered using the prediction errors $e(i,j)$ and prediction values $\hat{p}(i,j)$, which are calculated by Equation (4), Equation (5) and Equation (6). Finally, the original image can be recovered completely.

When a receiver has both the encryption key and the data hider key, he/she can recover both the original image and embedded data.

(1) Yi *et al.'* s method [30]:

Original bit-streams: 0000000000110100 (16 bits)

n = 16, n_a = 2, m = 3 > n_a, Bad block

Compressed bit-streams: 000000000000110100 (length = 18)

(2) Chen *et al.'* s method [3]:

Original bit-streams: 0000000000110100 (16 bits)

$s = 4$, $T = 3$, $1 \leqslant z = 3 \leqslant T$, G-II block

Block label → $(10)_2$, $\lceil \log_2 T \rceil$ bits to store z → $(11)_2$

First '1': 0000000000110100 ($n = 16$)

$z_1 = 11$, $q_1 = \lceil \log_2 n \rceil = 4$ bits, $t_1 = z_1 = 11$ → $(1011)_2$

Second '1': 0000000000110100 ($n - z_1 = 5$)

$z_2 = 12$, $q_2 = \lceil \log_2 (n - z_1) \rceil = 3$ bits, $t_2 = z_2 - z_1 = 1$ → $(001)_2$

Third '1': 0000000000110100 ($n - z_2 = 4$)

$z_3 = 14$, $q_3 = \lceil \log_2 (n - z_2) \rceil = 2$ bits, $t_3 = z_3 - z_2 = 2$ → $(10)_2$

Compressed bit-streams: 101110110010 (*length* = 13)

(3) Yin *et al.'* s method [32]: Type label 00, $L_{fix} = 4$

Original bit-streams: 0000000000011100 (16 bits)

First run: 00000000000 (11 bits) (⩾ 4 bits)

$l = \lfloor \log_2 11 \rfloor = 3$, $L_{mid} = (11\text{-}2^3)_2 = (011)_2$

$L_{pre} = 110$, $L_{mid} = 011$, $L_{tai} = 0$ → $(1100110)_2$

Second run: 111 (3 bits) (< 4 bits)

$L_{pre} = 0$, $L_{mid} = 1110$ → $(01110)_2$

Third run: 0 (1 bit) (< 4 bits)

$L_{pre} = 0$, $L_{mid} = 0$ → $(00)_2$

Compressed bit-streams: 110011001110 00 (*length* = 14)

(4) Arithmetic coding method [21]:

Original bit-streams: 0000000000110100 (16 bits)

Compressed bit-streams: 0001101010011000 (*length* = 16)

(5) Our BDBE method:

Original bit-streams: 0000000000110100 (16 bits)

Detail in Fig. 2.(d) Code: 01111010100, sign: 1

Compressed bit-streams: 011110101001 (*length* = 12)

Fig. 6. Comparisons of five encoding methods on a binary block.

4 EXPERIENCE RESULTS

4.1 The Performance of BDBE

To show the superiority of our BDBE. We compare it with four classical and state-of-the-art distance encoding methods, including Yi et al.'s [30], Chen et al.'s [3], Yin et al.'s [32], and binary arithmetic coding [21] methods. Notice that the encoding method in Yin et al.'s [32] method is an extended run-length encoding method by choosing the best scanning order from the four scanning order to a binary block.

4.1.1 Encoding Processes of These Methods. We first compare the encoding concepts of these encoding methods and Figure 6 shows the encoding processes of these methods to a 4×4 binary block. For Yi et al.'s [30] method, the threshold of the number of 1s is $n_a = 2$ for block size $s = 4$. Thus the block with three 1s cannot be encoded. Besides, it needs a block label '00' to mark this kind of block. Then a total number of 18 bits are required to represent this block. For Chen et al.'s [3] method, the threshold of the number of 1s is $T = 3$ for block size $s = 4$. Then the block is a G-II block with block label '10'. The final code includes the block label, number of 1s and the index codes of all the 1s. It is '1011101100110' with 13 bits. For Yin et al.'s [32] method, the best block scanning order is the type '00' after trying all the four order types. After encoding, the block code is '11001100111000' with 14 bits. For the binary arithmetic coding [21], the block can be encoded using 16 bits without counting additional symbol proportion information for recovery. For our BDBE method, the encoding process for the block is shown in Figure 2(d), and only 12 bis are required to encode the block. From Figure 6, it can be observed that, since our BDBE method can encode the distance from both sides using the least bits, it can use less bits to present a binary sequence.

4.1.2 Comparisons on Natural Images. To better show the superiority of our BDBE method, we compare it with the four encoding methods on six natural images. All the test images are greyscale

(a) (b) (c) (d) (e) (f)

Fig. 7. Six classical images with size 512×512. (a) *Baboon*; (b) *Boat*; (c) *Jetplane*; (d) *Lena*; (e) *Man*; (f) *Peppers*.

images with size 512×512, shown in Figure 7. For each test image, its pixel $p(i, j)$ can be decomposed into 8 bits, $(b_8b_7b_6b_5\ b_4b_3b_2b_1)_2$, where b_8 and b_1 are the most and least significant bits, respectively. These bits can be derived as follows:

$$b_k = \left\lfloor \frac{p(i,j)}{2^{k-1}} \right\rfloor \mod 2,\ k = 1, 2, \ldots, 8. \tag{11}$$

Thus we can get the 8 bit planes of each test image.

For our BDBE, we set the block size as $S = 4$ and the detailed encoding process for each bit plane is as follows. First, encode each block as a bitstream **BC** and a sign ξ. Second, connect the code **BC** of all the blocks to obtain a bitstream C_1 and connect the sign ξ of all the Type-II blocks to get a bitstream S_1. Due to the common occurrence of the same ξ sign within consecutive Type-II blocks in a bit plane, the bitstream S_1 exhibits high redundancy and can be further compressed. Third, encode S_1 using the BDBE method and obtain two bitstreams C_2 and S_2, which is the same as the first two steps. Thus, the final codes of a bit plane includes C_1, C_2, and S_2. During recovery process, S_1 is first reconstructed from C_2 and S_2, followed by the recovery of the original bit plane from C_1 and S_1.

Table 5 lists the encoding results of Yi et al.'s [30], Chen et al.'s [3], and Yin et al.'s [32] method, binary arithmetic coding [21] and our BDBE methods. Since most of the methods encode bit planes with block size 4×4, it is fair to fix block size to 4×4 and set the rest of parameters according to the original literature. As evidenced, our BDBE consistently achieves a better compression performance than other encoding methods on natural images.

4.2 The Performance of BDBE-RDHEI

In this section, we simulate the proposed BDBE-RDHEI, evaluate its performance and compare it with other RDH-EI schemes. Six classical images shown in Figure 7 and two image databases "BOWS-2" and "BOSSbase" containing 10,000 images are used as the test images. All the test images are greyscale images with size 512×512.

4.2.1 Simulation Results. Figure 8 simulates the process of our BDBE-RDHEI scheme for image *Jetplane*. Figure 8(b)–(i) show the 8 least significant bit planes of the error image **EI**. It can be seen that there are more 0s in the higher bit plane, which can obtain better encoding result. After encoded by the BDBE method, 4 bit-planes of the **EI** shown in Figure 8(h) and (i) are un-encodable bit-planes, and the other bit-planes are encodable bit-planes. After encrypting the encoding results using the AES, and embedding some random bits into the vacated room, we can obtain the encrypted image **E**, which is shown in Figure 8(j). After embedded into some additional data, the marked encrypted image is also randomlike, which is shown in Figure 8(k). One cannot obtain any useful information about the original image and the embedded additional data. Using the marked encrypted image and the encryption key, one can completely recover the original image, shown in Figure 8(l).

Table 5. The Sizes of the Encoding Results of Different Methods on Six Classical Images

Bit plane	*Baboon*					*Boat*				
	[30]	[3]	[32]	[21]	BDBE	[30]	[3]	[32]	[21]	BDBE
8	169463	235691	173388	261639	164997	96166	237408	88264	235685	96044
7	215355	238616	228905	259754	206303	121194	128549	117753	198940	118578
6	256554	262144	262144	259299	243661	185368	208618	195399	251117	177812
5	262144	262144	262144	261735	262144	254312	262144	262144	262144	241483
4	262144	262144	262144	262144	262144	262144	262144	262144	261423	262144
3	262144	262144	262144	262144	262144	262144	262144	262144	261878	262144
2	262144	262144	262144	262144	262144	262144	262144	262144	262012	262144
1	262144	262144	262144	262144	262144	262144	262144	262144	261923	262144
Total bits	1952092	2047171	1975157	2091003	**1925681**	1705616	1885295	1712136	1995122	**1682493**
Bit plane	*Jetplane*					*Lena*				
	[30]	[3]	[32]	[21]	BDBE	[30]	[3]	[32]	[21]	BDBE
8	71165	262017	53246	181480	73985	76030	182014	67414	262072	83216
7	110252	262144	102278	191728	108771	115526	177971	117456	257335	118534
6	132036	138401	126597	208357	128870	154380	218618	166839	262132	153339
5	183636	245523	191895	260634	176575	220065	254579	241194	262144	212158
4	233654	262144	249856	261895	222836	262144	262144	262144	262144	256635
3	262144	262144	262144	262144	260004	262144	262144	262144	262144	262144
2	262144	262144	262144	262144	262144	262144	262144	262144	262144	262144
1	262144	262144	262144	262144	262144	262144	262144	262144	262144	262144
Total bits	1517175	1956661	1510304	1890526	**1495329**	1614577	1881758	1641479	2092259	**1610314**
Bit plane	*Man*					*Peppers*				
	[30]	[3]	[32]	[21]	BDBE	[30]	[3]	[32]	[21]	BDBE
8	79626	116208	69904	218387	82958	67219	153018	53784	258350	73808
7	123782	147498	125313	227423	123733	114526	147949	114877	234900	115183
6	188715	208609	202037	251535	182514	163216	212581	176335	260272	160979
5	237386	240981	255613	255163	226117	224758	249249	245995	261410	215994
4	262144	262144	262144	259073	255188	262144	262144	262144	261947	262144
3	262144	262144	262144	261561	262144	262144	262144	262144	262142	262144
2	262144	262144	262144	261245	262144	262144	262144	262144	262144	262144
1	262144	262144	262144	262114	262144	262144	262144	262144	262144	262144
Total bits	1678085	1761872	1701443	1996501	**1656942**	1618295	1811373	1639567	2063309	**1614540**

In addition, our scheme is applicable to color images, such as RGB images, and cannot cause color distortion. When processing a color image, our scheme treats each color channel as a greyscale image, and is individually applied to each color channel. During the encryption phase, each color channel is encrypted into randomlike data, as demonstrated in Figure 8(j). Then the encrypted color image becomes randomlike. This randomization ensures the security of the encrypted image. In the image recovery phase, the recovery process for each color channel is lossless. Thus, our method can recover the original color image without any color distortion.

4.2.2 Embedding Rate. The embedding capacity indicates that how many additional data can be embedded into the encrypted image by a data hider. The embedding rate can be tested by the

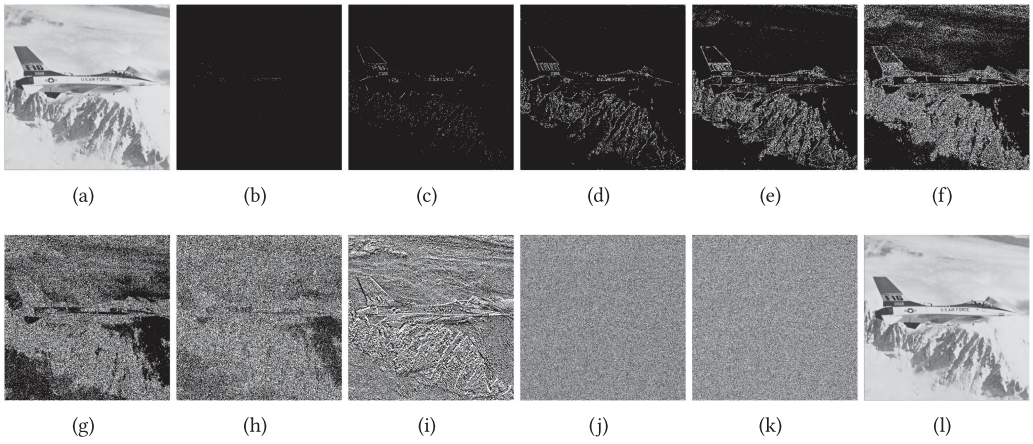

Fig. 8. Simulation results of the BDBE-RDHEI for image *Jetplane*. (a) Original image *Jetplane*; (b) the eighth bit plane of the error image **EI**; (c) the seventh bit plane of **EI**; (d) the sixth bit plane of **EI**; (e) the fifth bit plane of **EI**; (f) the fourth bit plane of **EI**; (g) the third bit plane of **EI**; (h) the second bit plane of **EI**; (i) the first bit plane of **EI**; (j) the encrypted image **E**; (k) the marked encrypted image **EM** with additional data embedded ($ER = 3.864$ *bpp*); (l) the reconstructed image (PSNR $\rightarrow +\infty$).

maximum embedded bits per pixel *bpp*, which is calculated as

$$ER = \frac{\text{Total embedding capacity} - \text{Payload}}{\text{Total number of pixels}}, \tag{12}$$

where the payload is the side information **SI**. According to the image encoding in Section 3.2, the length of the side information **SI** can be obtained as

$$L_{SI} = 41 + 3k + (19 - k) \times \lceil \log_2(MN) \rceil. \tag{13}$$

The total embedding capacity is the difference between the original image bits and the encrypted bitstream **EMI**. The **EMI** and the main information **MI** have the same length and $\mathbf{MI} = \mathbf{T}||\mathbf{C}||\mathbf{U}$. It is obvious that the length of **U** is $kMN/2$. Suppose that the lengths of **T** and **C** are L_T and L_C, respectively, and the image is with size $M \times N$. Then we can calculate the total embedding capacity as

$$L = 8MN - L_{EMI} = 8MN - \lceil (L_T + L_C + kMN/2)/128 \rceil \times 128. \tag{14}$$

Thus, the ER can be obtained as

$$ER = \frac{L - L_{SI}}{MN} = 8 - \frac{L_{SI} + L_{EMI}}{MN}. \tag{15}$$

We take the image *Lena* as an example. After calculating the prediction errors, we find that there is one pixel, whose prediction error is without $[-128, 127]$. We store the pixel in the binary sequence T with $L_T = 1 \times 8 = 8$, and a location map is used to indicate its position. Besides, the location map is the ninth bit plane of the error image. Calculate the proportion of 1s P_1 in each bit-plane, and determine the block size S of each bit-plane according to Equation (9), where $S = 1$ means it is an un-encodable bit-plane. Table 6 shows the encoded results of the 18 bit-planes by BDBE.

It can be seen bit-planes 13 to 18 are un-encodable bit-planes, and we can obtain $k = 6$. Thus, from Equation (13), we get $L_{SI} = 293$. To show the details of encoding **EI**, Table 6 lists the number of each type of blocks. It can be seen that in most encodable bit-planes, the number of the blocks with $n_1 = 0$ are the most prevalent. Then, the total length of the encoding results of the first 12

Table 6. The Encoding Results of the Error Image of *Lena*

Bit-plane ID	P_1	Block size S	Number of block with different types			Length of code
			I	II ($n_1 = 0$)	II ($n_1 > 0$)	
1	7.63×10^{-6}	256	0	1	1	21
2	0	256	0	2	0	4
3	4.58×10^{-5}	256	0	1	1	88
4	1.53×10^{-4}	128	0	5	3	293
5	2.96×10^{-3}	32	0	85	43	3940
6	5.91×10^{-3}	16	2	401	109	7118
7	2.01×10^{-2}	8	78	1530	440	18653
8	3.43×10^{-2}	8	198	1415	435	26379
9	8.26×10^{-2}	8	481	710	857	52306
10	9.88×10^{-2}	4	1506	4274	2412	58735
11	2.30×10^{-1}	4	3916	690	3586	108186
12	2.42×10^{-1}	4	4291	926	2975	108508
13	3.73×10^{-1}	1	—	—	—	131072
14	3.76×10^{-1}	1	—	—	—	131072
15	4.39×10^{-1}	1	—	—	—	131072
16	4.42×10^{-1}	1	—	—	—	131072
17	4.90×10^{-1}	1	—	—	—	131072
18	4.87×10^{-1}	1	—	—	—	131072

Table 7. The Embedding Rates of the Six Test Images

Test images	Parameter k	L_{SI}	L_T	L_C	L_{EMI}	ER (bpp)
Baboon	9	248	32	454,357	1,634,048	1.766
Boat	7	278	56	420,168	1,337,728	2.896
Jetplane	4	323	8	559,645	1,084,032	3.864
Lena	6	293	8	384,231	1,170,688	3.533
Man	7	278	192	458,114	1,375,872	2.750
Peppers	6	293	0	464,428	1,250,944	3.227

bit-planes is 384231, indicating that $L_C = 384231$. According to Equation (14), we can determine $L_{EMI} = 1170688$, and then calculate that $ER = 3.533$ bpp from Equation (15).

Table 7 lists the number of the un-encodable bit-planes k, the length of side information L_{SI}, the binary sequence $\mathbf{T}$'s length L_T, the total length of the encodable bit-planes' encoding results L_C, the length of the encrypted main information L_{EMI}, and ER of the six test images. As can be seen from Table 7, different images have different embedding rates, since they have different features. For example, the ER of the image *Jetplane* can be 2.18 times that of the image *Baboon*, because the the image *Jetplane* is much smoother than the image *Baboon*.

Table 8 shows the best, worst and average embedding rates of the two databases, "BOWS-2" and "BOSSbase." For the "BOSSbase" database, the best case is the 5162.*pgm* with $ER = 7.796$ bpp, and the worst case is the image 1377.*pgm* with $ER = 1.028$ bpp. For the "BOWS-2" database, the best case is the image 1478.*pgm* with $ER = 7.196$ bpp, and the worst case is the image 6184.*pgm* with $ER = 0.920$ bpp. The average embedding rate value of "BOSSbase" can reach to 4 bpp. Since the receiver can completely recover the original image using the encryption key, the recovered image is lossless and then the PSNR $\rightarrow +\infty$ and SSIM = 1.

Table 8. The Best, Worst, and Average Embedding Rates of Two Databases in our BDBE-RDHEI

Performance	BOSSbase	BOWS-2
Best	7.796	7.196
Worst	1.028	0.920
Average	4.103	3.996
PSNR	$+\infty$	$+\infty$
SSIM	1	1

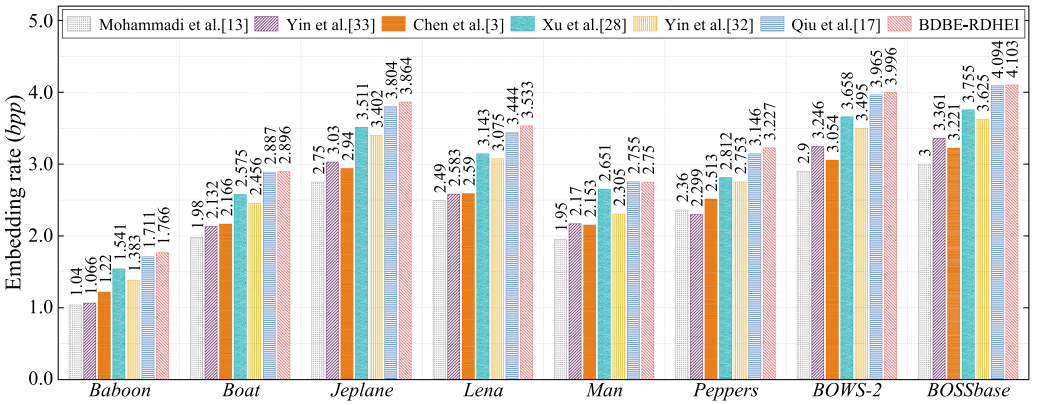

Fig. 9. Embedding rate comparisons of different RDH-EI schemes on the six test images and two databases.

4.2.3 Comparisons of Embedding Rate. To show the high embedding rate of our proposed BDBE-RDHEI, we compare it with several state-of-the-art RDH-EI schemes, including Mohammadi et al.'s [13], Yin et al.'s [33], Chen et al.'s [3], Xu et al.'s [28], Yin et al.'s [32], and Qiu et al.'s [17] schemes. Figure 9 shows the comparison results of the embedding rate over the six classical images and the images in two databases, in which the embedding rates of the two databases are the average values. The results show that our BDBE-RDHEI scheme can achieve the largest embedding rate. Take the image *Lena* as example, the embedding rate of our BDBE-RDHEI method is 3.533 *bpp*, while that of Mohammadi et al.'s [13], Yin et al.'s [33], Chen et al.'s [3], Xu et al.'s [28], Yin et al.'s [32], and Qiu et al.'s [17] schemes are 2.49 *bpp*, 2.583 *bpp*, 2.59 *bpp*, 3.143 *bpp*, 3.075 *bpp*, and 3.444 *bpp*, respectively. Besides, for the two databases, the average embedding rates of our BDBE-RDHEI scheme are larger than that of Qiu et al.'s [17] scheme, which achieves the best results among these RDH-EI schemes.

4.2.4 Security Analysis. In our BDBE-RDHEI scheme, the encoded images are encrypted using the AES, while the data to be embedded can be encrypted using any existing encryption methods. Both the original image and embedded data can achieve a high security. To show the high security of protecting the original image, we investigate the correlation of images before and after encryption.

Since a natural image has high correlations between adjacent pixels, a pixel can reveal the information of its surrounding pixels and attackers can use this feature to predict the image information. To avoid this statistical attack, the strong correlations of adjacent pixels should be broken in an encrypted image. We first plot the adjacent pixel pairs of the natural image and its encrypted image by our scheme in Figure 10. As can be seen, most of the pixel pairs of the plain image along the horizontal, vertical and diagonal directions are distributed on the diagonal line of the 2D planes,

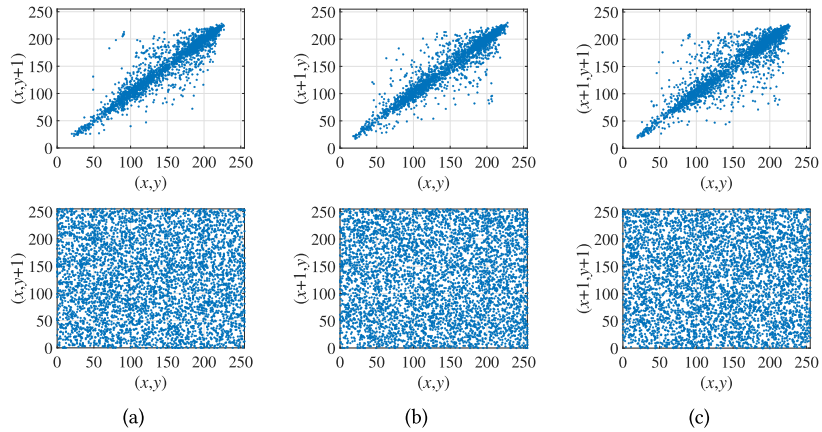

Fig. 10. The first and second rows are the adjacent pixel pair plots of the original image *Jetplane* and its encrypted image along the (a) horizontal direction, (b) vertical direction, and (c) diagonal direction, respectively.

which indicates that a pixel of the plain image has high correlations with its adjacent pixels. However, the pixel pairs of the encrypted image along the three directions are distributed randomly on the 2D planes. This indicates that a pixel has low correlations with its adjacent pixels.

We also use the **correlation coefficient (CC)** to calculate the adjacent pixel correlations. The CC is calculated as

$$CC_{\mathbf{u},\mathbf{v}} = \frac{\sum_{i=1}^{N_s}(u_i - \bar{u})(v_i - \bar{v})}{\sqrt{\sum_{i=1}^{N_s}(u_i - \bar{u})^2} \times \sqrt{\sum_{i=1}^{N_s}(v_i - \bar{v})^2}}, \tag{16}$$

where $\bar{u}$ and $\bar{v}$ are the average values of the vector $\mathbf{u}$ and $\mathbf{v}$, respectively, and $\mathbf{u}$ and $\mathbf{v}$ are two vectors formed by pixel pairs along the horizontal, vertical, or diagonal directions. The CC values of some test images are listed in Table 9. It shows that the CC values of the plain images are close to 1, which indicates high correlations of the two vectors [1]. However, the CC values of the encrypted and marked encrypted images are close to 0, which means low correlations [8]. Besides, we also calculate the information entropy of these images, and the results listed in Table 9 demonstrate that the pixels in the encrypted and marked encrypted images are close to the theoretically maximum value 8, indicating that these images have uniform-distribution pixels to defense statistic attack.

4.2.5 Complexity Analysis. In our BDBE–RDHEI scheme, the complexity of room reservation is $O(MN)$, including the generation and compression of the error image **EI**. The compression of **EI** involves a maximum of three traversals, including the calculation P_1, determining the block size S and encoding each block. We test the running time of seven RRBE-based RDH-EI schemes on the image *Lena* with an embedding rate 1 *bpp*. All the schemes are implemented using the MATLAB programming language and run on 64-bit Windows 11 with AMD Ryzen 7 5800H @3.20 GHz, 16 GB RAM. Table 10 depicts the results. During image encryption process, our scheme and the schemes in References [33] and [28] exhibit fast processing speeds. In contrast, the schemes in References [13], [3], [17], and [32] are time-consuming due to various factors, such as the use of AES for random byte generation, the iterative prediction method, excessive reliance on arithmetic encoding, and multiple compression of a bit plane with various rearrangements to optimize the result. Notably, our BDBE-RDHEI scheme outperforms Qiu et al.'s [17] scheme, which achieves the best embedding rates among prior RDH-EI schemes.

Table 9. Correlation Coefficient and Information Entropy Scores of Some Original, Encrypted and Marked Encrypted Images

Test images	Image types	Correlation coefficient values			Information entropy
		Horizontal	Vertical	Diagonal	
Baboon	Original image	0.8685	0.7500	0.7333	7.3583
	Encrypted image	−0.0078	0.0127	−0.0205	7.9993
	Marked encrypted image	0.0190	0.0049	0.0146	7.9976
Boat	Original image	0.9402	0.9726	0.9221	7.1914
	Encrypted image	0.0165	−0.0021	−0.0021	7.9993
	Marked encrypted image	0.0059	0.0078	0.0232	7.9961
Jetplane	Original image	0.9674	0.9656	0.9410	6.7025
	Encrypted image	0.0206	−0.0119	0.0116	7.9992
	Marked encrypted image	0.0100	0.0088	−0.0199	7.9950
Lena	Original image	0.9712	0.9851	0.9557	7.4451
	Encrypted image	−0.0049	0.0015	−0.0066	7.9993
	Marked encrypted image	−0.0045	0.0111	−0.0140	7.9955
Man	Original image	0.9627	0.9696	0.9469	7.3574
	Encrypted image	0.0141	−0.0043	−0.0026	7.9993
	Marked encrypted image	0.0031	0.0272	0.0129	7.9963
Peppers	Original image	0.9806	0.9815	0.9681	7.5715
	Encrypted image	−0.0230	0.0038	0.0153	7.9994
	Marked encrypted image	0.0092	0.0144	−0.0208	7.9958

Table 10. Running Time Comparison (in Seconds) of Different RRBE-Based RDH-EI Schemes for Image *Lena*

Parties	Operation	Methods						
		Mohammadi et al. [13]	Yin et al. [33]	Chen et al. [3]	Xu et al. [28]	Yin et al. [32]	Qiu et al. [17]	BDBE-RDHEI
Content owner	Image Encryption	1.501	0.697	6.157	0.814	11.685	5.115	0.868
Data hider	Data Embedding	1.878	0.441	0.458	0.450	0.831	1.605	0.469
Receiver	Data Extraction	1.231	0.724	0.365	0.443	0.704	1.463	0.455
	Image Recovery	3.812	1.645	8.899	0.746	8.513	1.982	1.194

5 CONCLUSION

In this article, we first propose a BDBE method. It encodes the distances of values in binary sequence from both sides, and thus the distances can be represented using least bits. Using the BDBE, we further develop an RDH-EI scheme. It allows the content owner to preserve embedding room for data embedding. After being encrypted using the AES, the encrypted image codes are sent to data hider for data embedding. A receiver can extract the embedded data and recover the original image using the related keys. We implement a comprehensive evaluation and the evaluation results show that our scheme ensures image content confidentiality, while achieving a large embedding capacity. Comparison results demonstrate that it can achieve larger embedding capacity than state-of-the-art methods. Our BDBE method employs a fixed block size for each bit-plane, but the compression rate can be further improved by using varying block sizes within the bit-plane, as different image regions exhibit different data distributions. Our future work aims to enhance the embedding capacity through efficient compression of image prediction errors.

REFERENCES

[1] Jacob Benesty, Jingdong Chen, and Yiteng Huang. 2008. On the importance of the Pearson correlation coefficient in noise reduction. *IEEE Trans. Audio Speech Lang. Process.* 16, 4 (2008), 757–765.

[2] Rupali Bhardwaj and Ashutosh Aggarwal. 2020. An improved block based joint reversible data hiding in encrypted images by symmetric cryptosystem. *Pattern Recognit. Lett.* 139 (2020), 60–68.

[3] Fan Chen, Yuan Yuan, Hongjie He, Miao Tian, and Heng-Ming Tai. 2021. Multi-MSB compression based reversible data hiding scheme in encrypted images. *IEEE Trans. Circ. Syst. Video Technol.* 31, 3 (2021), 905–916.

[4] Kaimeng Chen and Chin-Chen Chang. 2019. High-capacity reversible data hiding in encrypted images based on extended run-length coding and block-based MSB plane rearrangement. *J. Vis. Commun. Image Represent.* 58 (2019), 334–344.

[5] Bo Guan and Dawen Xu. 2020. An efficient high-capacity reversible data hiding scheme for encrypted images. *J. Vis. Commun. Image Represent.* 66 (2020), 102744.

[6] Yan Ke, Minqing Zhang, Xinpeng Zhanga, Jia Liu, Tingting Su, and Xiaoyuan Yang. 2022. A reversible data hiding scheme in encrypted domain for secret image sharing based on chinese remainder theorem. *IEEE Trans. Circ. Syst. Video Technol.* 32, 4 (2022), 2469–2481.

[7] Fouad Khelifi. 2018. On the security of a stream cipher in reversible data hiding schemes operating in the encrypted domain. *Sign. Process.* 143 (2018), 336–345.

[8] Liang Kong and Heng Nian. 2021. Fault detection and location method for mesh-type DC microgrid using pearson correlation coefficient. *IEEE Trans. Power Del.* 36, 3 (2021), 1428–1439.

[9] Qi Li, Bin Yan, Hui Li, and Na Chen. 2018. Separable reversible data hiding in encrypted images with improved security and capacity. *Multimedia Tools Appl.* 77 (2018), 30749–30768.

[10] Zilong Liu and Chi-Man Pun. 2022. Reversible data hiding in encrypted images using chunk encryption and redundancy matrix representation. *IEEE Trans. Depend. Secure Comput.* 19, 2 (2022), 1382–1394.

[11] Kede Ma, Weiming Zhang, Xianfeng Zhao, Nenghai Yu, and Fenghua Li. 2013. Reversible data hiding in encrypted images by reserving room before encryption. *IEEE Trans. Inf. Forens. Secur.* 8, 3 (2013), 553–562.

[12] Mehedi Masud, Ghulam Muhammad, Hesham Alhumyani, Sultan S. Alshamrani, Omar Cheikhrouhou, Saleh Ibrahim, and M. Shamim Hossain. 2020. Deep learning-based intelligent face recognition in IoT-cloud environment. *Computer Commun.* 152 (2020), 215–222.

[13] Ammar Mohammadi, Mansor Nakhkash, and Mohammad Ali Akhaee. 2020. A high-capacity reversible data hiding in encrypted images employing local difference predictor. *IEEE Trans. Circ. Syst. Video Technol.* 30, 8 (2020), 2366–2376.

[14] W. Puech, M. Chaumont, and O. Strauss. 2008. A reversible data hiding method for encrypted images. In *Security, Forensics, Steganography, and Watermarking of Multimedia Contents X*, Vol. 6819. International Society for Optics and Photonics, 534 – 542.

[15] Pauline Puteaux and William Puech. 2021. A recursive reversible data hiding in encrypted images method with a very high payload. *IEEE Trans. Multimedia* 23 (2021), 636–650.

[16] Zhenxing Qian and Xinpeng Zhang. 2016. Reversible data hiding in encrypted images with distributed source encoding. *IEEE Trans. Circ. Syst. Video Technol.* 26, 4 (2016), 636–646.

[17] Yingqiang Qiu, Qichao Ying, Yuyan Yang, Huanqiang Zeng, Sheng Li, and Zhenxing Qian. 2022. High-capacity framework for reversible data hiding in encrypted image using pixel prediction and entropy encoding. *IEEE Trans. Circ. Syst. Video Technol.* 32, 9 (2022), 5874–5887.

[18] Lingfeng Qu, Fan Chen, Shanjun Zhang, and Hongjie He. 2021. Cryptanalysis of reversible data hiding in encrypted images by block permutation and co-modulation. *IEEE Trans. Multimedia* 24 (2021), 2924–2937.

[19] Meng Shen, Guohua Cheng, Liehuang Zhu, Xiaojiang Du, and Jiankun Hu. 2020. Content-based multi-source encrypted image retrieval in clouds with privacy preservation. *Fut. Gener. Comput. Syst.* 109 (2020), 621–632.

[20] Shaowei Weng, Caiying Zhang, Tiancong Zhang, and Kaimeng Chen. 2021. High capacity reversible data hiding in encrypted images using SIBRW and GCC. *J. Vis. Commun. Image Represent.* 75 (2021), 102932.

[21] Ian H. Witten, Radford M. Neal, and John G. Cleary. 1987. Arithmetic coding for data compression. *Commun. ACM* 30, 6 (1987), 520–540.

[22] Hao-Tian Wu, Yiu ming Cheung, Zhiyuan Yang, and Shaohua Tang. 2019. A high-capacity reversible data hiding method for homomorphic encrypted images. *J. Vis. Commun. Image Represent.* 62 (2019), 87–96.

[23] Xiaolin Wu and Nasir Memon. 1997. Context-based, adaptive, lossless image coding. *IEEE Trans. Commun.* 45, 4 (1997), 437–444.

[24] Zebin Wu, Jin Sun, Yi Zhang, Yaoqin Zhu, Jun Li, Antonio Plaza, JÃşn Atli Benediktsson, and Zhihui Wei. 2021. Scheduling-guided automatic processing of massive hyperspectral image classification on cloud computing architectures. *IEEE Trans. Cybern.* 51, 7 (2021), 3588–3601.

[25] Zhihua Xia, Qiuju Ji, Qi Gu, Chengsheng Yuan, and Fengjun Xiao. 2022. A format-compatible searchable encryption scheme for JPEG images using bag-of-words. *ACM Trans. Multimedia Comput. Commun. Appl.* 18, 3, Article 85 (March 2022), 18 pages.
[26] Zhihua Xia, Leqi Jiang, Dandan Liu, Lihua Lu, and Byeungwoo Jeon. 2019. BOEW: A content-based image retrieval scheme using bag-of-encrypted-words in cloud computing. *IEEE Trans. Serv. Comput.* 15, 1 (2019), 202–214.
[27] Lizhi Xiong, Xiao Han, Ching-Nung Yang, and Zhihua Xia. 2023. RDH-DES: Reversible data hiding over distributed encrypted-image servers based on secret sharing. *ACM Trans. Multimedia Comput. Commun. Appl.* 19, 1, Article 5 (January 2023), 19 pages.
[28] Shuying Xu, Ji-Hwei Horng, Ching-Chun Chang, and Chin-Chen Chang. 2023. Reversible data hiding with hierarchical block variable length coding for cloud security. *IEEE Trans. Depend. Secure Comput.* 20, 5 (2023), 4199–4213.
[29] Shuang Yi and Yicong Zhou. 2015. An improved reversible data hiding in encrypted images. In *Proceedings of the IEEE China Summit and International Conference on Signal and Information Processing (ChinaSIP '15).* 225–229.
[30] Shuang Yi and Yicong Zhou. 2017. Binary-block embedding for reversible data hiding in encrypted images. *Sign. Process.* 133 (2017), 40–51.
[31] Shuang Yi and Yicong Zhou. 2019. Separable and reversible data hiding in encrypted images using parametric binary tree labeling. *IEEE Trans. Multimedia* 21, 1 (2019), 51–64.
[32] Zhaoxia Yin, Yinyin Peng, and Youzhi Xiang. 2022. Reversible data hiding in encrypted images based on pixel prediction and bit-plane compression. *IEEE Trans. Depend. Secure Comput.* 19, 2 (2022), 992–1002.
[33] Zhaoxia Yin, Youzhi Xiang, and Xinpeng Zhang. 2020. Reversible data hiding in encrypted images based on multi-MSB prediction and huffman coding. *IEEE Trans. Multimedia* 22, 4 (2020), 874–884.
[34] Weiming Zhang, Kede Ma, and Nenghai Yu. 2014. Reversibility improved data hiding in encrypted images. *Sign. Process.* 94 (2014), 118–127.
[35] Xinpeng Zhang. 2011. Reversible data hiding in encrypted image. *IEEE Sign. Process. Lett.* 18, 4 (2011), 255–258.
[36] Jiantao Zhou, Weiwei Sun, Li Dong, Xianming Liu, Oscar C. Au, and Yuan Yan Tang. 2016. Secure reversible image data hiding over encrypted domain via key modulation. *IEEE Trans. Circ. Syst. Video Technol.* 26, 3 (2016), 441–452.

Received 26 June 2023; revised 1 November 2023; accepted 17 December 2023