11th International Conference on Information Science and Technology (ICIST)
Chengdu, China, May 21-23, 2021

2021 11th International Conference on Information Science and Technology (ICIST) | 978-1-6654-1266-7/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICIST52614.2021.9440649

Reversible data hiding method in encrypted images using secret sharing and Huffman coding

Shuang Yi*
Engineering Research Center of Forensic Science of Chongqing Education Committee
College of Criminal Investigation
Southwest University of Political Science and Law
Chongqing 401120, China

Juan Zhou
College of Criminal Investigation
Southwest University of Political Science and Law
Chongqing 401120, China

Zhongyun Hua
School of Computer Science and Technology
Harbin Institute of Technology
Shenzhen 518000, China

Yong Xiang
School of Information Technology
Deakin University
Victoria 3125, Australia

***Abstract*—Early works on reversible data hiding in encrypted images (RDHEI) usually generate only one encrypted image from the original image, if the encrypted image is lost or maliciously damaged by an attacker, all information about the original image will be lost. Thus, storing images in distributed servers is an effective solution. Therefore, this paper proposes an RDHEI method using (k, n)-threshold secret sharing. It is not a traditional secret sharing technique, but takes its principles and ideas. In this algorithm, we combine the image encryption and sharing process to generate n shares and send them to the cloud for storage. It maintains the advantage of (k, n)-threshold secret sharing technique that only at least k shares can successfully recover the original image while less than k shares cannot. In addition, the size of each share is smaller than the original image, so that it can save storage spaces. By Huffman coding the pixel difference in the encrypted image block, secret data are embedded into these shares. It is a full reversible method that data extraction and image recovery are performed separately and losslessly. Simulation results, comparisons and security analysis are demonstrated to show superior performances of the proposed algorithm.**

***Index Terms*—reversible data hiding, secret image sharing, sharing matrix, image encryption**

Reversible data hiding (RDH) [1] in digital images is a technique that can losslessly recover the cover image after extracting the embedded secret data. It generates the data-embedded-image visually indistinguishable from the cover image and aims to protect the secret data from being revealed. Recently, a lot of RDH methods have been proposed, such as histogram shifting [2]–[4], difference expansion [5], [6], prediction-error expansion [7], [8], etc. Image encryption is another popular technique for privacy protection [9], [10]. It changes the meaningful image into noise-like one so that the unauthorized user can not access its content. With the popularization of mobile Internet technology and the rise and wide application of cloud computing technology, the mode of storing, editing and processing data on the local PC has gradually changed to the mode of providing services on the cloud computing platform [11], [12]. To protect users' privacy, images are encrypted before uploading to the cloud. The cloud service provider may add tags (hereafter we call it secret data) to the encrypted image to indicate the source, type, owner, authentication information and so on, for ease of management. The receiver only with corresponding keys can extract secret data and/or recover the original image. Thus, the reversible data hiding technology in encrypted images (RDHEI) is developed and it can realize the safe storage and management of cover images.

Existing RDHEI methods mainly use the block cipher [13], stream cipher or homomorphic encryption to encrypt the original image. For example, method in [13] uses the block cipher to encrypt the original image and embed one bit of secret data into each encrypted image block, data extraction is performed by analyzing the local standard deviation of each decrypted block. Inspired by this method, Zhang [14] proposed an RDHEI method by applying stream cipher to each image block, data embedding is accomplished by flipping the 3 least significant bits(LSBs) of each encrypted image blocks. Later, some improvements of RDHEI methods have been proposed, such as reducing the data extraction error by side-match [15], improving the embedding rate by compression [16], key modulation [17], histogram shifting [18], [19], prediction-error expansion [20], room reservation [21]–[25]. Most of these previous methods are separable RDHEI methods, which means data extraction and image recovery are performed separately. On the other hand, some researchers show their interests in developing RDHEI methods using homomorphic encryption [26]–[30], such as

This work was supported in part by the National Natural Science Foundation of China under Grant 62002301, the Natural Science Foundation of Chongqing under Grant cstc2019jcyj-msxmX0393 and cstc2018jscx-msybX0115, the Education Committee foundation of Chongqing under Grant KJQN201900305, KJQN201900307 and KJQN201900310, Research Foundation of Southwest University of Political Science and Law under Grant 2018XZQN-32, the Research Start-up Foundation of SWUPL.
*Corresponding author: Shuang Yi: yishuang@swupl.edu.cn

Paillier encryption [31]. However, the computation cost of these methods is extremely high and the generated encrypted image usually suffers from serious data expansion, which means the bit-depth of the encrypted image is much larger than the original one.

Early works on RDHEI usually generate only one encrypted image from the original image, if the encrypted image is lost or maliciously damaged by an attacker, all information about the original image will be lost. Therefore, in order to make the application loss-tolerant, a better solution is to store the image in distributed cloud-based servers. However, if each cloud server holds a full copy of the encrypted image, it will cause a waste of storage space. In addition, attackers will have more chances to get the encrypted image from multiple servers. Thus, we suggest to use the k-out-of-n threshold secret sharing method to solve these problems. The definition of k-out-of-n threshold secret sharing was proposed by Shamir [31], where n different noise-like shares (shadows) are generated from the original image and any no less than k shares $(k < n)$ can recover the original image, while less than k shares can not. Then, each share is stored in different cloud servers. All shares are noise-like images so that the k-out-of-n threshold secret sharing can also be regarded as an encryption method [32]. Fig. 1 shows the application model of cloud-based secret sharing and data hiding in encrypted domain. There are three entities: content-owner, cloud data center(CDC) and receiver. The content-owner divides the original image into several noise-like shares and sends them to different CDCs. The CDC then embeds secret data into the share. Once receiving the share transmission request, the CDC extracts the secret data from the share and sends the share to the receiver. After obtaining sufficient shares, the receiver is able to recover the original image losslessly. In this way, even if $n-k$ CDCs are compromised, the original image can still be recovered from other k shares.

Recently, Wu *et al.* [30] proposed an RDHEI method using the Shamir's (k, n)-threshold secret sharing. It uses the polynomials over a finite filed to generate shares and constructs the polynomial by *Lagrange Interpolation* to obtain the original image. In this method, n shares are generated using a pair-wise image encryption method and each share has the same size as the original image. Therefore, for an original image with Z pixels, the generated n shares contain a total of $n * Z$ pixels. Because adjacent pixels are encrypted by same parameters, pixel pairs in each share preserve the difference as it in the original image, traditional RDH methods such as difference expansion and difference histogram shifting can be used for data embedding. Due the utilization of prime number 251 in an 8-bit pixel image, this algorithm needs a preprocessing to restrict the pixel value in the range of [0, 250] before image sharing.

In order to realize the distributed storage and management of digital images, this paper proposes a reversible data hiding method using (k, n)-threshold secret sharing. It is not a sharing technique in the traditional sense, but takes its principles and ideas. The main contributions and advantages of the proposed

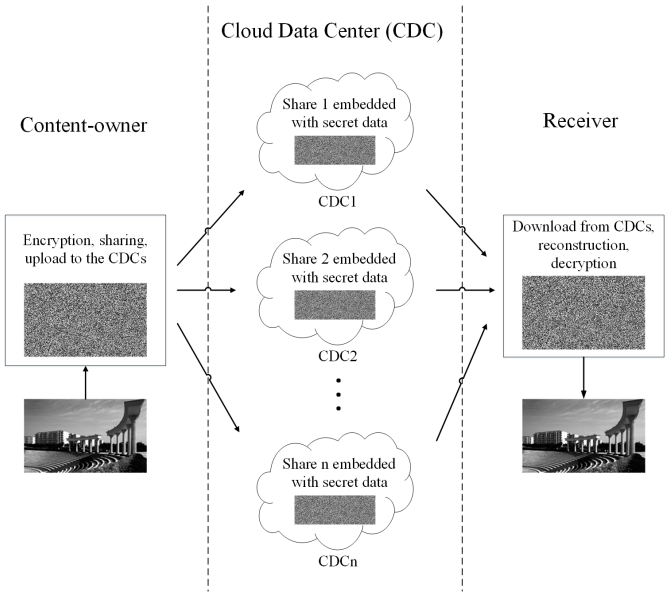


Fig. 1: Application model of cloud-based secret sharing and data hiding in encrypted domain.

algorithm are summarized as follows.

- We proposed an RDHEI method for distributed storage and management. It combines the encryption and (k, n)-sharing matrix to generate n shares and uses the Huffman coding for data hiding.
- The size if each share is smaller than the original image, therefore, it can save storage spaces.
- The loss of less than $n - k$ shares will not affect the recovering of the original image, which makes the application more fault-tolerant. Meanwhile, any fake share and/or the lack of enough true shares will cause failing to recover the original image, which makes the application model more secure.
- Experimental results and analysis demonstrate the proposed algorithm has relatively higher embedding rate and better performances than several related work.

The rest of this paper is organized as follows. Section I reviews the (k, n)-sharing matrix definition [33] and (k, n)-sharing matrix generation method [34]. The RDHEI method using (k, n)-threshold secret sharing is proposed in Section II. Section III demonstrates the experimental results of the proposed algorithm and comparisons with several existing works. The security is analyzed in Section IV. Finally, we draw the conclusion in Section V.

I. Preliminary

In this section, we first review the definition of (k, n)-sharing matrix and secret sharing/reconstruction processes using the sharing matrix [33]. Then a (k, n)-sharing matrix generation method [34] is reviewed and followed by a discussion to show that the generated matrix satisfies the (k, n)-sharing matrix definition in [33].

A. (k,n)-sharing matrix definition

Let $S^{(k,n)}$ be an $n \times w$ binary matrix, $S^{(k,n)}(i,j) \in \{0,1\}$, where $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant w$. Randomly selecting any p rows of elements from matrix $S^{(k,n)}$ generates a $p \times w$ binary matrix $Z(s,j)$, where $1 \leqslant p \leqslant n$ and $1 \leqslant s \leqslant p$. If matrix $S^{(k,n)}$ satisfies three following three conditions, it is called the (k,n)-sharing matrix. 1)

1) there is at least one "1" in each row in matrix $S^{(k,n)}$, namely

$$\sum_{j=1}^{w} S^{(k,n)}(i,j) \neq 0 \tag{1}$$

2) there is at least one "1" in each column in matrix Z when $p \geqslant k$, namely

$$\sum_{s=1}^{p} Z(s,j) \neq 0 \tag{2}$$

3) there is at least one zero column in matrix Z when $p < k$, namely

$$\prod_{j=1}^{w} \left(\sum_{s=1}^{p} Z(s,j) \right) = 0 \tag{3}$$

For example: Eq. (4) is a $(3,4)$-sharing matrix satisfying the conditions in Eqs. (1)-(3).

$$S^{(3,4)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

B. Secret sharing using (k,n)-sharing matrix

1) Sharing process: Suppose that we want to share the original integer data P into n shares using $S^{(k,n)}$, where P and $S^{(k,n)}$ are with a size of $1 \times w$ and $n \times w$, respectively. We construct the i^{th} share $R_{(i)}$ by

$$R_{(i)} = P * S(i,:), \quad i = 1, 2, \cdots, n \tag{5}$$

where "*" is the point-to-point multiplication in this paper. Therefore, the i^{th} share $R_{(i)}$ is generated by the i^{th} row of the sharing matrix $S^{(k,n)}$.

2) Reconstruction process: When k_r ($k_r \geqslant k$) shares are combined, we can losslessly reconstruct the original data P. Firstly, we combine these k_r shares to form a matrix $R_{[k_r]}$ with a size of $k_r \times w$, where each row of $R_{[k_r]}$ is a share. Similarly, we construct a matrix $S_{[k_r]}$ which contains the corresponding k_r rows of the original sharing matrix $S^{(k,n)}$. Thus, $S_{[k_r]}$ is with a size of $k_r \times w$ as well. Then, we reconstruct the original data P by

$$P(j) = R_{[k_r]}(1,j) || R_{[k_r]}(2,j) ||, \cdots, || R_{[k_r]}(k_r,j) \tag{6}$$

where $j = 1, 2, \cdots, w$ and "||" is the bit-level boolean function "or".

3) Example: Here, we use an example to show the detailed secret sharing and reconstruction processes using the sharing matrix. Suppose the secret data $P = [1\ 2\ 3\ 4\ 5\ 6]$ and the sharing matrix $S^{(3,4)}$ in Eq. (4) is used to distribute P into 4 shares. Using Eq. (5), we can obtain shares $R_{(1)} = [0\ 2\ 0\ 0\ 5\ 6]$, $R_{(2)} = [0\ 0\ 3\ 4\ 0\ 6]$, $R_{(3)} = [1\ 0\ 0\ 4\ 5\ 0]$ and $R_{(4)} = [1\ 2\ 3\ 0\ 0\ 0]$. Then, we randomly select k_r ($k_r \geqslant 3$) shares to form $R_{[K_r]}$. E.g., if $k_r = 3$ and $R_{(1)}$, $R_{(2)}$ and $R_{(4)}$ are selected, we obtain $R_{[3]}$ as

$$R_{[3]} = \begin{bmatrix} 0 & 2 & 0 & 0 & 5 & 6 \\ 0 & 0 & 3 & 4 & 0 & 6 \\ 1 & 2 & 3 & 0 & 0 & 0 \end{bmatrix} \tag{7}$$

Using Eq. (6), we obtain the recovered $P = [1\ 2\ 3\ 4\ 5\ 6]$.

C. (k,n)-sharing matrix generation

In [34], Chao *et al.* introduced a (k,n) shadows-assignment matrix. In this paper, we call it (k,n)-sharing matrix. Next, we review the generation method of the (k,n)-sharing matrix and followed by an illustrative example.

1) Generation method: Firstly, we construct a matrix M_1 with a size of $n \times 1$. It consists of $(n-k+1)$ ones and $(k-1)$ zeros, where $1 < k < n$. For example, $M_1 = [1\ 1\ 1\ 0\ 0]^T$ for $(k,n) = (3,5)$. Then, we obtain all possible permutations of M_1, denoted as M_i, $i = 2, 3, ...\mathcal{N}$, where $\mathcal{N} = C_n^{k-1} = \frac{n!}{(k-1)!(n-k+1)!}$. Finally, we concatenate these $\mathcal{N}$ matrices to form the (k,n)-sharing matrix $S^{(k,n)}$ as shown in Eq. (8), where $S^{(k,n)}$ is with a size of $n \times \mathcal{N}$.

$$S^{(k,n)} = [M_1, M_2, ..., M_{\mathcal{N}}] \tag{8}$$

2) An illustrative example: Here, we give an example of generation of sharing matrix $S^{(3,5)}$. We first generate matrices $M_1 = [1\ 1\ 1\ 0\ 0]^T$ and obtain its all possible permutations M_i ($i = 2, 3, ..., 10$). Finally, we concatenate M_i ($i = 1, 2, ..., 10$) to obtain the sharing matrix $S^{(3,5)}$ as shown in Eq. (9)

$$M_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix},\ M_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix},\ M_3 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix},\ M_4 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix},\ M_5 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$M_6 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix},\ M_7 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix},\ M_8 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix},\ M_9 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix},\ M_{10} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$S^{(3,5)} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{9}$$

D. Discussion

Here, we discuss that the generated sharing matrix satisfies the three requirements of (k,n)-sharing matrix definition.

- According to the sharing matrix generation algorithm, each column of $S^{(k,n)}$ is a permutation of M_1 which contains $(n-k+1)$ ones and $(k-1)$ zeros, and $S^{(k,n)}$ is formed by all permutations of M_1. Therefore,

each row of $S^{(k,n)}$ contains $\frac{(n-k+1)\mathcal{N}}{n}$ ones. Because $\frac{(n-k+1)\mathcal{N}}{n} = \frac{(n-1)!}{(k-1)!(n-k)!} \geqslant 1$, there is at least one "1" in each row of $S^{(k,n)}$ which meets the condition in Eq. (1).

- We randomly select p rows from $S^{(k,n)}$ to form matrix Z, where each column of $S^{(k,n)}$ contains $(n-k+1)$ ones. When $p \geqslant k$, each column of Z contains at least $p-k+1 \geqslant 1$ ones. Thus, there is at least one "1" in each column when $p \geqslant k$ which perfectly meets the second condition of Eq. (2).
- We randomly select p rows from $S^{(k,n)}$ to form matrix Z, where each column of $S^{(k,n)}$ contains $(k-1)$ zeros. If $p < k$, we obtain $p \leqslant k-1$. Thus, there exists at least one "0" column in Z when $p < k$, and the third condition is satisfied as well.

II. Proposed algorithm

The framework of the proposed algorithm is shown in Fig. 2. It consists of three phases: 1) generation of shares at the content-owner side; 2) data embedding and extraction at the service-provider side; 3) image recovery at the receiver side. In the first phase, the original image is encrypted and separated into n shares. Then these shares are sent to different cloud service-providers for secret data embedding. Once receive the image transmission request, the service-provider extracts secret data from marked shares and then sends them to the receiver. At the receiver side, after obtaining no less than k shares, the receiver reconstructs these shares to form a single image and decrypt it losslessly to get the recovered image. Next, we introduce our proposed algorithm in detail.

A. Generation of shares

Generation of shares including two steps: 1) image encryption and 2) sharing process. It first generates an encrypted image with same size of original image, and then separate it into n shares using the (k,n)-sharing matrix, where the size of each share is smaller than the original image.

1) Image encryption: Assume that an $m_1 \times m_2$ original image $\mathbb{I}$, with pixel values in the range of [0, 255], is divided into t non-overlapped blocks. Each block $\mathbb{I}_{i,j}$ is with a size of 2×2, where (i,j) is the row and column index of each block, thus $t = m_1 * m_2/4$ and $1 \leqslant i \leqslant m_1/2$, $1 \leqslant j \leqslant m_2/2$. We use $\mathbb{I}^p_{i,j}$ to denote the p^{th} pixel of the block $\mathbb{I}_{i,j}$ in raster-scan order, where $p = 1,2,3,4$. The framework of image encryption method is shown in Fig. 3. We first apply a block permutation to shuffle all blocks within the original image, then use a row and column substitution operation to change pixel values. In order to enhance the security, four rounds of permutation and substitution procedures are performed. Next, we introduce the image encryption method in detail.

According to the image encryption key K_{enc}, we shuffling the image block within the original image. Note that the each image block is considered as a single unit for permutation. The permutation operation does not rely on any specific method, any scrambling algorithm can be used. In block substitution procedure, we first generate one row and one column of 2×2 random blocks R_j and C_i using K_{enc}, and add them to the top and left of the original image, respectively, where i and j are block index, pixel values in the same random block are equal and all values are in the range of [0, 255]. Fig. 4 shows an example of adding surrounding pixels to an original image with 25 blocks. Using K_{enc}, we then apply the row and column substitution operation using Eqs. (10) and (11), respectively, where $V_{i,j}$ and $W_{i,j}$ are random values within the range of [0, 255]. These two equations show one round of block substitution where $\mathbb{I}$ and $\mathbb{O}$ are the input and output images, respectively. Then the output image is considered as the new input one to do the second round of block permutation and substitution. After four rounds, we obtain the final encrypted image $\mathbb{E}$. Note that the random values C_i, R_j, $V_{i,j}$ and $W_{i,j}$ used in each round are different and they all generated by the image encryption key K_{enc}.

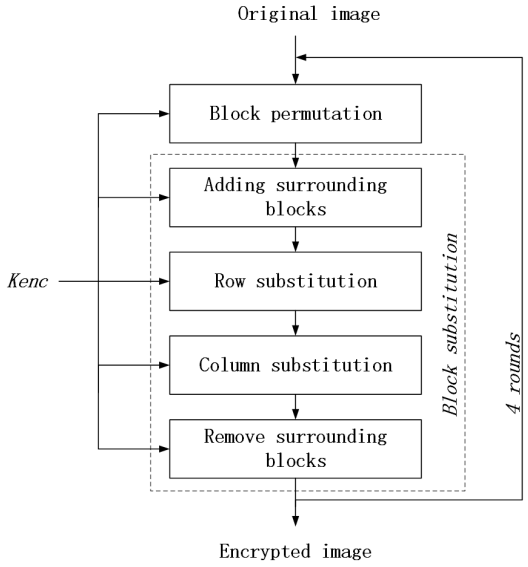


Fig. 3: Framework of image encryption.

	R_1	R_2	R_3	R_4	R_5
C_1	$I_{1,1}$	$I_{1,2}$	...		
C_2	$I_{2,1}$				
C_3	...				
C_4					...
C_5				...	$I_{5,5}$

Fig. 4: An example of adding surrounding pixels.

$$\mathbb{U}^p_{i,j} = \begin{cases} (\mathbb{I}^p_{i,j} + \sum_{k=1}^{4} C^k_i + V_{i,j}) \mod 256, & (\text{if } j = 1) \\ (\mathbb{I}^p_{i,j} + \sum_{k=1}^{4} \mathbb{U}^k_{i,j-1} + V_{i,j}) \mod 256, & (\text{otherwise}) \end{cases} \tag{10}$$

$$\mathbb{O}^p_{i,j} = \begin{cases} (\mathbb{U}^p_{i,j} + \sum_{k=1}^{4} R^k_i + W_{i,j}) \mod 256, & (\text{if } i = 1) \\ (\mathbb{U}^p_{i,j} + \sum_{k=1}^{4} \mathbb{O}^k_{i-1,j} + W_{i,j}) \mod 256, & (\text{otherwise}) \end{cases} \tag{11}$$

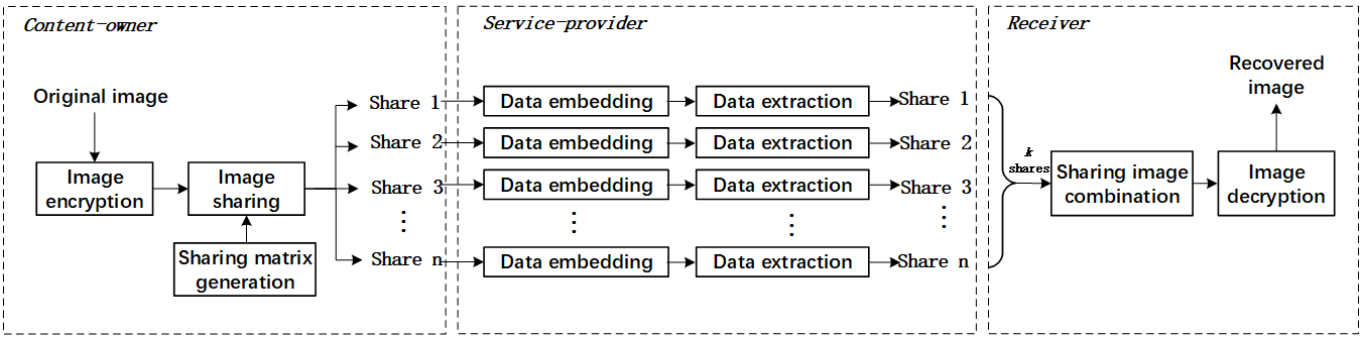


Fig. 2: Framework of the proposed algorithm.

2) Sharing process: After obtaining the encrypted image, we generate a sharing matrix $\mathcal{S}^{(k,n)}$ for secret image sharing based on parameters k and n defined by the content-owner. In this phase, each 2×2 image block is considered as a single unit for processing. Because there are t image blocks, the size of $\mathcal{S}^{(k,n)}$ should be $n \times t$. Then, we explain how to generate $\mathcal{S}^{(k,n)}$ and n shares.

According to the sharing matrix generation method shown in Section I-C, we first generate a basic sharing matrix $S^{(k,n)}$ with a size of $n \times \mathcal{N}$. If $\mathcal{N} \geqslant t$, we then randomly select t columns from $S^{(k,n)}$ using K_{enc} to generate the matrix $\mathcal{S}^{(k,n)}$; otherwise, we duplicate $S^{(k,n)}$ for $\lceil \frac{t}{\mathcal{N}} \rceil$ times to form a large matrix with a size of $n \times \lceil \frac{t}{\mathcal{N}} \rceil \mathcal{N}$. Using K_{enc}, we permute all columns in $\mathcal{S}^{(k,n)}$ and then randomly select t columns from this matrix to generate $\mathcal{S}^{(k,n)}$. Note that the permutation only changes the column indexes while keeps the row indexes unmodified.

For image sharing, the encrypted image should be first reshaped into a 1D block matrix with a size of $1 \times t$ blocks, where each element is a 2×2 encrypted image block. Therefore, the 1D block matrix is actually with a size of $2 \times 2t$ pixels. Using $\mathcal{S}^{(k,n)}$ and the image sharing method proposed in Section I, we then generate n shares, where each share is also a 1D block matrix, and the number of blocks equals to the number of "1"s in the corresponding row matrix of $\mathcal{S}^{(k,n)}$. For example, if the i^{th} row of $\mathcal{S}^{(k,n)}$ contains q_i number of "1"s, the i^{th} share contains q_i image blocks. For ease of illustration, we reshape each share into an image with a size of $m_1 \times \hat{m}_2$, where $\hat{m}_2 = \lceil \frac{4q_i}{m_1} \rceil$. Note that, each 2×2 block is considered as a single unit for processing. If $4q_i \mod m_1 \neq 0$, we randomly select $\frac{m_1}{2} - (q_i \mod \frac{m_1}{2})$ blocks from the current share and append them to the end of the share.

In order to successfully reconstruct the encrypted image from these shares, we should know the original image size, sharing matrix and the mapping relationship between the share and sharing matrix's row index. Because the row sizes of the original image and the shares are equal, we only need to store the collum value m_2. For the sharing matrix, we store the parameters k and n. Therefore, we use 40 bits to store these information, where 16 bits for storing the collum value m_2 and 8 bits for storing the parameters k, n and sharing matrix's row index i, respectively. Because the redundancy is preserved in the image blocks, we can embed the 40-bit information into the share by traditional RDH methods. In this paper, we adopt the DE method [35] for embedding. For example, using the security key K_{enc}, we can randomly select some image blocks for embedding. Finally, these shares are upload to n different CDCs for storage.

Huffuman code	Source symbol	Probability
0	A1	0.4
100	A2	0.21
101	A3	0.15
110	A4	0.13
111	A5	0.11

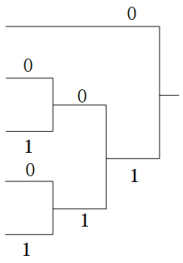


Fig. 5: Example of Huffman coding.

B. Data embedding and extraction

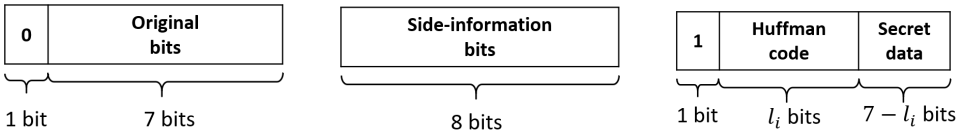
1) Data embedding: At the service-provider side, some additional data are embedded into the encrypted shares for the purpose of management. The data embedding procedure consists of three steps: 1) pixel grouping; 2) Huffman coding; 3) data embedding.

1) Pixel grouping

For each image block in share $\mathbb{R}$, we calculate the difference value e_i^j by

$$e_i^j = \mathbb{R}_i^1 - \mathbb{R}_i^j, \quad (j = 2, 3, 4) \tag{12}$$

where i is the block index and we assume that there are t_s blocks in $\mathbb{R}$. We then separate whole pixels in $\mathbb{R}_i$ into three categories, namely reference pixel, embeddable pixel and non-embeddable pixel. Reference pixels consist of the first pixel $\mathbb{R}_i^1$ in each block, and they will not be changed during the data embedding phase. For the rest pixels, we separate them into two categories based on the difference value and a given positive integer γ. If $-\gamma \leqslant e_i^j \leqslant \gamma$, the current pixel is embeddable pixel; otherwise, it is non-embeddable pixel. The embeddable pixel is able to embed additional data while the non-embeddable pixel is not. The value of parameter γ



(a) Non-embeddable pixel in group-I (b) Non-embeddable pixel in group-II (c) Embeddable pixel

Fig. 6: Illustration of bits allocation for data embedding.

will influence the embedding rate and it will be discussed in Section III. Once γ is given, the number of embeddable pixels b_e and non-embeddable pixels b_n are calculated, and thus $b_e + b_n = 3t_s$.

2) Huffman coding

Huffman coding is a method of compressing source symbols with variable-length codes based on the probabilities of source symbols. Symbols with high probability will assign short code lengths, while symbols with low probability will assign long code lengths. An example of Huffman coding is shown in Fig. 5. The source symbols A1, A2, A3, A4 and A5 are with the probabilities of 0.4, 0.21, 0.15, 0.13 and 0.11, respectively. After Huffman coding, symbols A1∼A5 are encoded with different lengths of code bits. The symbol A1 with the highest probability is encoded by a one-bit code "0", and the symbol A5 with the lowest probability is encoded by a three-bit code "111". Thus, the average code length of using Huffman coding is 1*0.4+3*0.21+3*0.15+3*0.13+3*0.11=2.2 bits. If we use fixed-length codes to encode these 6 symbols, the average code length should be $\lceil \log_2 6 \rceil = 3$ bits. Therefore, the utilization of Huffman coding can decrease the storage space. Once the source symbols' probabilities are given, the corresponding Huffman codes are generated.

3) Secret data embedding

In this phase, we compress embeddable pixels using Huffman coding to empty out spare space for secret data embedding. The detailed bits allocation for data embedding is shown in Fig. 6. Firstly, all non-embeddable pixels are separated into two categories, namely group-I and group-II with b_{n_1} and b_{n_2} pixels respectively, where pixels in group-II are randomly selected from the non-embeddable pixels using the data-hiding key K_d. Thus, $b_{n_1} + b_{n_2} = b_n$ and the value of b_{n_2} will be discussed later. Secondly, one bit is utilized to label the embeddable pixel and the non-embeddable pixel in group-I. Here we use "0" to denote the non-embeddable pixel in group-I and "1" to denote the embeddable pixel respectively. Because pixels in group-II are determined by K_d, we don't need any specific bit to label them. Thirdly, for the pixel in group-I, the remaining 7 bits will be kept unmodified; for the embeddable pixel, the rest bits will be replaced by Huffman code and secret data. The pixels in group-II are utilized for storing side-information. Next, we will introduce the data embedding procedure in detail.

In order to embed secret data, the embeddable pixels are compressed by Huffman coding. Here we compress the difference value e_i^j of embeddable pixels instead of directly compressing their pixel values. Because the value of e_i^j is in the range of $[-\gamma, \gamma]$, there are $2\gamma + 1$ different values. We encode all difference values by Huffman coding and totally get $2\gamma+1$ Huffman codewords, which are one-to-one mapping to the corresponding difference value. Here we denote the Huffman codewords by $c_{-\gamma}, c_{-\gamma+1}, \cdots, c_0, \cdots, c_\gamma$, where c_i $(-\gamma \leqslant i \leqslant \gamma)$ indicates the codeword when the difference value equals to i. We use l_i $(-\gamma \leqslant i \leqslant \gamma)$ to denote the bit length of the codeword c_i, thus the maximum value of l_i should be no larger than 7. For the embeddable pixel, according to e_i^j, we then replace following l_i bits by Huffman codeword and use the rest $7 - l_i$ bits to accommodate secret data. Because the non-embeddable pixel bits that have been replaced, including the first bit of pixels in group-I and all bits in group-II, are useful for image recovery at the receiver side, they should be stored with secret data. Therefore, the total number of pure embedded secret data bits T can be calculated by

$$T = \sum_{i=-\gamma}^{\gamma} (7 - l_i) p_i - b_{n_1} - 8 b_{n_2}, \tag{13}$$

where p_i is the number of embeddable pixels when the difference value equals to i. In order to enhance the security, the secret data should be encrypted before data embedding.

Since the Huffman code is needed in data extraction phase, the Huffman code lookup table should be stored in the share as the side-information. We use the pixels in group-II to embed the side-information by bit replacement, and the length of the side-information will be experimentally shown in Section III. After secret data and side-information embedding, we finally obtain the marked share $\mathbb{M}$.

2) Data extraction: After receiving the image transmission request, the service-provider extracts the secret data from marked shares and sends them to the receiver.

Because the data extraction procedures for each marked share are the same, we take one case for example. For the marked share $\mathbb{M}$, we first divide it into a number of 2×2 non-overlapped blocks. According to the labeling bits and K_d, we then classify the three bottom right pixels of each block into embeddable pixels, non-embeddable pixels of group-I and group-II, respectively. Next, we extract the side-information from the pixels in group-II. According to the Huffman code lookup table, we can easily identify the Huffman code and secret data. Thus secret data bits are extracted from the embeddable pixels sequentially. Finally, we

decrypt the extracted secret data bits using K_d to obtain the plain-text data.

To recover the share, after extracting the side-information, we obtain the difference value e_i^j of each embeddable pixels using the Huffman code lookup table and the codewords stored in embeddable pixels. Then, we recover the embeddable pixels by

$$\mathbb{M}_i^j = \mathbb{M}_i^1 - e_i^j. \tag{14}$$

For bits in non-embeddable pixels (group-I and group-II) that have been replaced, we use the extracted side-information to recover them. Therefore, all pixels in the sharing image are successfully recovered.

C. Image recovery

Image recovery consists of two steps: share combination and image decryption. Share combination process generates a single image (encrypted version) from obtained shares, image decryption process then decrypts it to obtain the original image.

1) Share combination: We denote the obtained k_r shares as $\mathbb{M}_{(1)}, \mathbb{M}_{(2)}, \cdots, \mathbb{M}_{(k_r)}$. For each share, we first extract the image column value m_2, sharing matrix parameters k, n, and the row index i of the sharing matrix using K_{enc}. According to the extracted information and K_{enc}, we generate the sharing matrix $\mathbb{S}^{(k,n)}$ with a size of $n \times t$ in the same way as proposed in Section II-A2. According to the row index i, we use the i^{th} row of $\mathbb{S}^{(k,n)}$ to recover the image share $\mathbb{M}_{(i)}$. We denote the i^{th} row of $\mathbb{S}^{(k,n)}$ by $\mathbb{S}_i^{(k,n)}$. Therefore, each binary number of $\mathbb{S}_i^{(k,n)}$ indicates a 2×2 block in recovered share. Therefore, we scan $\mathbb{S}_i^{(k,n)}$ bit by bit to recover the share. If the current bit is "1", we keep the corresponding block of $\mathbb{M}_{(i)}$ unmodified; if the current bit is "0", we insert a 2×2 zero block into the $\mathbb{M}_{(i)}$. Followed by these steps, all k_r shares are recovered and each of them is with a size of $2 \times 2t$ pixels. Next, we combine these k_r shares into a whole share $\mathbb{M}_{[k_r]}$ with a size of $2k_r \times 2t$ pixels. Finally, we reconstruct the all image blocks using Eq. (6). Here an image block is considered as a single element for processing. Finally, we reshape the reconstructed image blocks into an $m_1 \times m_2$ image $\mathbb{E}$, which is the same as the encrypted image. The reshape way is the inverse procedure as proposed in Section II-A2.

D. Image decryption

Image decryption is an inverse procedure of encryption. According to K_{enc}, we first add surrounding blocks to $\mathbb{E}$ and apply the column and row block substitution using Eqs. (15) and (16), respectively, where $\mathbb{E}$ and $\mathbb{U}$ are the input and output images for one round of block substitution. We then inversely permutate the blocks and using the generated image as the input image to do the second round of inverse substitution and permutation. All the random values used in the decryption phase are generated by the same way as proposed in encryption phase. After four rounds, we obtain the final decrypted image $\mathbb{I}$.

(b) (c) (d) (e)

Fig. 7: The four Test images. (a) *Lena*; (b) *Baboon*; (c) *Boat*; (d) *Goldhill*

$$\mathbb{O}_{i,j}^p = \begin{cases} (\mathbb{E}_{i,j}^p - \sum_{k=1}^4 R_i^k - W_{i,j}) \mod 256, & (\text{if } i = 1) \\ (\mathbb{E}_{i,j}^p - \sum_{k=1}^4 \mathbb{E}_{i-1,j}^k - W_{i,j}) \mod 256, & (\text{otherwise}) \end{cases} \tag{15}$$

$$\mathbb{U}_{i,j}^p = \begin{cases} (\mathbb{O}_{i,j}^p - \sum_{k=1}^4 C_i^k - V_{i,j}) \mod 256, & (\text{if } j = 1) \\ (\mathbb{O}_{i,j}^p - \sum_{k=1}^4 \mathbb{O}_{i,j-1}^k - V_{i,j}) \mod 256, & (\text{otherwise}) \end{cases} \tag{16}$$

III. Simulation results and comparisons

In this section, we show the experimental results of the proposed algorithm and the comparisons with some related work. Fig. 7 shows all test images used in the demonstration. All of them are 8-bit public available standard images[1] with a size of 512×512, including "*Lena*", "*Baboon*", "*Boat*" and "*Goldhill*".

A. Simulations

Figs. 8-9 show the experimental results of the proposed algorithm using (3, 5)-sharing matrix and (7, 8)-sharing matrix respectively. As can be seen, all shares and marked shares are noise-like images, and the original image can be perfectly recovered at the receiver side. In addition, the size of each share is smaller than that of the original image. Because each share discards some image blocks according to the "0"s in the sharing matrix. We use μ_1 and μ_n to calculate the data expansion rate of one share and n shares comparing to the original image, respectively, where μ_1 is defined as

$$\mu_1 = \frac{\lambda_s}{\lambda_o}, \tag{17}$$

λ_s and λ_o represent the size of a share and the original image, respectively. Using the proposed sharing matrix, μ_1 can be approximately calculated by

$$\mu_1 \approx \frac{n-k+1}{n}. \tag{18}$$

and thus

$$\mu_n = n\mu_1 \approx n - k + 1. \tag{19}$$

Therefore, the size of the share is $\frac{n-k+1}{n}$ of the original image, and thus the share's size in Fig. 9 and Fig. 8 are $\frac{3}{5}$ and $\frac{1}{4}$ of the original image, respectively.

[1] http://decsai.ugr.es/cvg/dbimagenes/g512.php

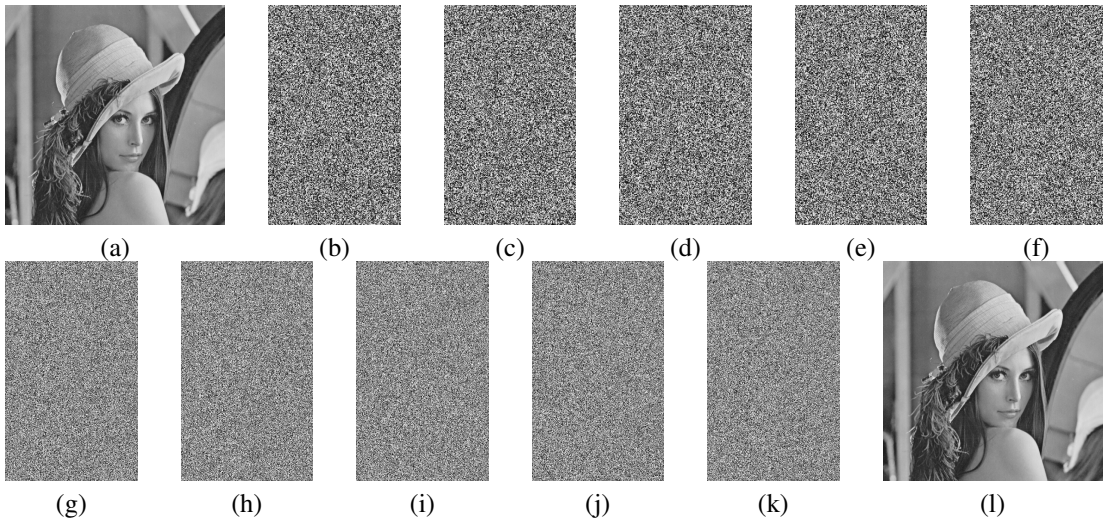


Fig. 8: Simulation results of the proposed algorithm using (3,5)-sharing matrix. (a) The original *Lena* image; (b)-(f) 5 shares; (g)-(k) 5 marked shares with embedding rate of (g) 1.6735 bpp, (h)1.6778 bpp, (i)1.6696 bpp, (j)1.6737 bpp, (k) 1.6725 bpp, respectively, where $\gamma = 13$; (l) the losslessly reconstructed image using any 3 marked shares.

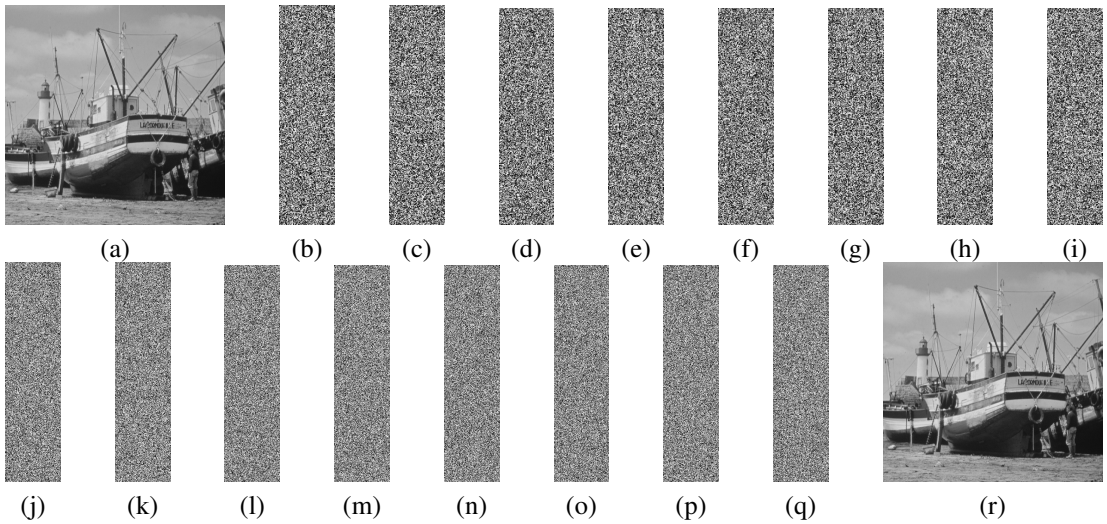


Fig. 9: Simulation results of the proposed algorithm using (7,8)-sharing matrix. (a) The original *Boat* image; (b)-(f) 8 shares; (j)-(q) 8 marked shares with embedding rate of (j) 1.5691 bpp, (k) 1.5715 bpp, (l) 1.5492 bpp, (m) 1.5614 bpp, (n) 1.5652 bpp, (o) 1.5701 bpp, (p) 1.5660 bpp, (q) 1.5709 bpp, respectively, where $\gamma = 13$; (r) the losslessly reconstructed image using any 7 marked shares.

Because the size and number of the shares are not equal to that of the original image, we introduce a new definition of the embedding rate (*ER*) by

$$ER_s = \frac{\text{total embedded bits in the } i^{th} \text{ share}}{\text{total bits of the } i^{th} \text{ share / 8}} \tag{20}$$

and

$$ER_a = \frac{\sum_{i=1}^{n} \text{total embedded bits in the } i^{th} \text{ share}}{\sum_{i=1}^{n} \text{total bits of the } i^{th} \text{ share / 8}} \tag{21}$$

where ER_s and ER_a indicate the embedding rate for each share and the average embedding rate for n shares, respectively. Usually, ER_s and ER_a have similar values, because image blocks are randomly distributed after image encryption and thus they are randomly distributed in each share as well. In Figs. 8 and 9, we set the parameter $\gamma = 13$ for demonstration, the average embedding rates ER_a of the experiments are 1.6734 bpp and 1.5654 bpp, respectively.

Using difference values of γ, the number of embeddable and non-embeddable pixels will be different, thus results in different embedding rates. We use these four test images to demonstrate how does the parameter γ affect the embedding rate. The results are listed in Tables I and II, where (k, n) are set to (3, 5) and (7, 8), respectively. From the results we can observe that, as the r increases, the average embedding rate ER_a increases first and then decreases. The maximum average embedding rate of the four test images are 1.688 bpp (*Lena*), 0.634 bpp (*Baboon*), 1.572 bpp (*Boat*) and 1.364 bpp (*Goldhill*), respectively. As can be seen, images with more

smooth texture will achieve a higher embedding rate. When r is small, it may result in negative embedding rate, which means it can not embed any secret data. For example, when $r = 1$, the average embedding rate for *Baboon* is -0.350 bpp. This is because most of the pixels are non-embeddable pixels, which result in more space to store the auxiliary information, including the first bit of pixels in group-I and all pixel bits in group-II, and thus the pure embedded bits T calculated by Eq. (13) will be negative. When r is large, it is unable to embed data well, because some of the Huffman codewords length may larger than 7. For example, $r \geqslant 14$ for *Lena* when (k, n) equals to (3,5) as shown in Table. I. In addition, compare the two tables we can see that, the average embedding rates for the same image are similar under various r. This is because the blocks in encrypted are randomly distributed, no matter what n is, image blocks in each share are randomly distributed as well.

Next, we discuss about the side-information that is made up of the Huffman code lookup table. Given a certain value n, each of the n shares has similar image size and texture smoothness. Therefore, the length of the side-information for each encrypted sharing image will be roughly equal. In this demonstration, we show the average side-information length of n encrypted sharing images. The results are listed in Tables III and IV, where (k, n) are set to (3, 5) and (7, 8), respectively. From the results we can observe that the length of the side-information increases as the increasing of r. This is because under a certain r, there are totally $2r + 1$ difference value e should be encoded by Huffuman coding, which results in $2r + 1$ Huffman codewords to be stored. In addition, the length of side-information is not very large so that they can be stored in dozens of pixels. When r is relatively large, e.g., $r \geqslant 14$ for *Lena* image, we don't list the side-information length, since the maximum length of the Huffman codeword is larger than 7. In addition, compare Tables III and IV we can see that for the same test image, the values of (k, n) have little effect on the length of the side-information. For example, when $r \leqslant 12$, the side-information for *Lena* image have the same length when (k, n) is set to (3, 5) and (7, 8); when $r = 13$, their side-information differs in length by only 1 bit. Because for a certain test image, each of n shares has similar image block statistical redundancy, results in similar histogram of difference value e and almost the same Huffman codes.

B. Comparisons

Different from most of the RDHEI methods that generate only one encrypted image, the proposed method generates multiple shares based on the parameters n. Therefore, we compare the data expansion rate of the proposed algorithm with several related methods. The results are shown in Table. V. Methods in [26] and [27] adopt the Paillier homomorphic encryption method, the encrypted image pixel will expand from 8 bits to 1024 bits when a 512-bit secret key is used. Therefore, the encrypted image is seriously expanded. Because methods in [17], [21], [26], [27], [29] generate only one encrypted image, the data expansion rate μ_1 for one encrypted image and μ_n for whole encrypted images are the same. Wu *et al.*'s method generates n shares of the same size as the original image. Therefore, the data expansion rate μ_n is n. For the proposed method, the size of each share is smaller than the original image due to the utilization of sharing matrix, the data expansion rate for one share and whole shares approximately equal to $(n-k+1)/n$ and $n-k+1$, respectively.

TABLE VI: Embedding rate comparison.

Method	Test images			
	Lena	*Baboon*	*Boat*	*Goldhill*
Li et al. [26]	$\leqslant$0.008	$\leqslant$0.008	$\leqslant$0.008	$\leqslant$0.008
Zhang et al. [27]	$\leqslant$ 0.008	$\leqslant$0.008	$\leqslant$0.008	$\leqslant$0.008
Zhou et al. [17]	0.188	0.188	0.188	0.188
Ma et al. [21]	0.952	0.683	0.996	0.912
Chen et al. [29]	0.500	0.500	0.500	0.500
Wu's DHSM [30]	0.647	0.562	0.626	0.645
Proposed	1.686	0.632	1.572	1.364

In order to show the embedding performances of the proposed algorithm, we compare it with several related works. The results are shown in Table VI. For fair of comparison, we use the following formula to calculate the embedding rate of method in [17], [21], [26], [27] . For Wu *et al.*'s and proposed method, we use the Eq. (21) for simulation. From the results we can observe that the proposed algorithm outperforms these existing methods in embedding capacity.

$$ER_e = \frac{\text{total embedded bits}}{\text{total bits of the encrypted image}/8} \tag{22}$$

IV. Security analysis

Using the proposed algorithm, the successful recovery of the original image depends on two conditions: enough true shares and correct key K_{enc}. Lack of any one condition can not recover the original image. For an unauthorized user, if he has the security key K_{enc}, he may collect true shares as much as he can and use the obtained insufficient shares to recover the original image. Alternatively, he may use a small amount of fake shares instead of true shares to recover the original image. Therefore, we should ensure that only enough true shares can successfully recover the original image. In addition, if someone has enough true shares, we should ensure the use of incorrect key can not reconstruct the original image as well.

Based on previous analysis, we verify the security of the proposed algorithm from the following three aspects: integrity analysis, fake share analysis and security key analysis.

A. Integrity analysis

For the (k, n)-secret sharing based RDHEI, we should ensure that the successful reconstruction of the original image depends on enough true shares. That is when $k_r < k$ or when fake shares are involved, the original image can not be successfully recovered. The simulation is shown in Fig. 10, where *Boat* image is applied for demonstration and $(k, n) = (5, 7)$. From the results in Fig. 10(b) we can see that

TABLE I: Average embedding rate ER_a of $\mathbb{S}^{(3,5)}$ based shares under various r.

ER_a (bpp)	r											
	1	3	5	7	9	10	12	13	14	16	19	22
Lena	0.466	1.228	1.548	1.656	**1.686**	**1.686**	1.678	1.672	—	—	—	—
Baboon	-0.350	-0.007	0.216	0.384	0.491	0.527	0.576	0.593	0.606	0.624	**0.632**	0.630
Boat	0.522	1.250	1.487	1.553	1.570	**1.572**	1.567	1.562	1.557	—	—	—
Goldhill	0.127	0.753	1.074	1.231	1.316	1.338	1.362	**1.364**	1.363	1.355	—	—

TABLE II: Average embedding rate ER_a of $\mathbb{S}^{(7,8)}$ based shares under various r.

ER_a (bpp)	r											
	1	3	5	7	9	10	12	13	14	16	19	22
Lena	0.468	1.231	1.552	1.660	**1.691**	**1.691**	1.683	1.677	—	—	—	—
Baboon	-0.349	-0.005	0.218	0.386	0.494	0.530	0.579	0.596	0.610	0.627	**0.635**	0.633
Boat	0.524	1.254	1.491	1.556	1.573	**1.575**	1.570	1.565	1.560	—	—	—
Goldhill	0.125	0.751	1.072	1.229	1.315	1.337	1.361	**1.363**	1.362	1.355	—	—

TABLE III: The average side-information length of shares using $\mathbb{S}^{(3,5)}$.

Side-information length (bits)	r											
	1	3	5	7	9	10	12	13	14	16	19	22
Lena	13	39	66	112	149	170	208	229	—	—	—	—
Baboon	13	39	66	94	144	162	198	218	237	275	333	394
Boat	13	39	73	114	151	169	210	232	252	—	—	—
Goldhill	13	39	66	111	145	164	201	222	241	282	—	—

TABLE IV: The average side-information length of shares using $\mathbb{S}^{(7,8)}$.

Side-information length (bits)	r											
	1	3	5	7	9	10	12	13	14	16	19	22
Lena	13	39	66	112	149	170	208	230	—	—	—	—
Baboon	13	39	66	94	144	162	199	218	237	275	333	395
Boat	13	39	75	114	151	169	210	232	253	—	—	—
Goldhill	13	39	66	111	145	164	201	222	241	283	—	—

when only 4 true shares are combined, the original image can not be successfully recovered.

B. Fake share analysis

Next, we demonstrate how fake shares affect the recovered image. Fig. 10(c) shows the result of image recovery using 4 true shares and 1 fake share. Fig. 10(d) shows the result of image recovery using 5 true shares and 1 fake share. As can be seen, even enough true shares are adopted, the reconstructed image is still noise-like when only 1 fake share is involved.

C. Security key analysis

Fig. 10(e) demonstrates the result of using 6 true shares and incorrect key K_{enc}. Therefore, even with enough true shares, it also can not recover the original image without the correct key. Only when the correct key is used and no less than 5 true shares are combined with no fake share, the original image can be completely recovered (Fig. 10(f)).

V. Conclusion

In this paper, a novel RDHEI method based on (k, n)-threshold secret sharing has been proposed. It encrypts the original image block by block with a size of 2×2 and uses a (k, n)-sharing matrix for image sharing. Each generated share has a smaller size than the original image. Experimental results and analysis have validated its efficacy and performances with several state-of-the-art approaches. In the future work, we are devote to developing sharing method to make the size of shares smaller so that it can save more storage spaces.

References

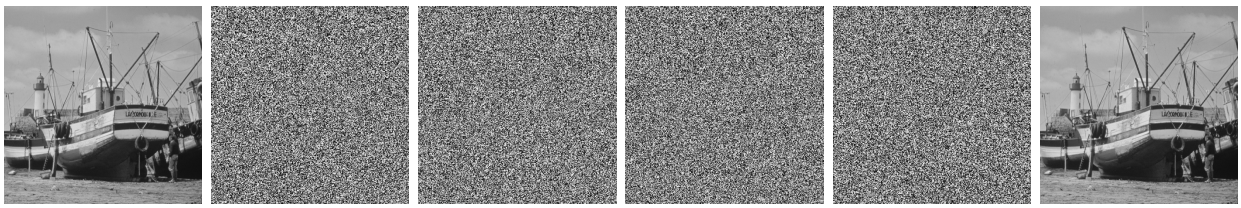
[1] Z. Ni, Y.-Q. Shi, N. Ansari, W. Su, Reversible data hiding, IEEE Transactions on Circuits and Systems for Video Technology 16 (3) (2006) 354–362.

[2] Y. J. Jia, Z. X. Yin, X. P. Zhang, Y. L. Luo, Reversible data hiding based on reducing invalid shifting of pixels in histogram shifting, Signal Processing 163 (2019) 238–246.

[3] X. Z. Xie, C. C. Chang, C. C. Lin, J. L. Lin, A turtle shell based rdh scheme with two-dimensional histogram shifting, Multimedia Tools and Applications 78 (14) (2019) 19413–19436.

TABLE V: Data expansion rate comparison.

Method	Bits of input pixel	Bits of marked encrypted pixel	Expansion rate μ_1	Expansion rate μ_n
Li et al. [26]	8	1024	128	128
Zhang et al. [27]	8	1024	128	128
Zhou et al. [17]	8	8	1	1
Ma et al. [21]	8	8	1	1
Chen et al. [29]	8	8	1	1
Wu et al. [30]	8	8	1	n
Proposed	8	8	$\approx (n-k+1)/n$	$\approx n-k+1$



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

Fig. 10: Security analysis of the proposed algorithm using *Boat* image with $(k, n) = (5, 7)$. (a) The original image; (b) the recovered image using 4 true shares; (c) the recovered image using 4 true shares and 1 fake share; (d) the recovered image using 5 true shares and 1 fake share; (e) the recovered image using 6 true shares and incorrect key K_{enc}; (f) the recovered image when any no less than 5 true shares are combined and the correct key is used.

[4] S. Kukreja, G. Kasana, S. S. Kasana, Adaptive reversible data hiding scheme for digital images based on histogram shifting, Computing and Informatics 38 (2) (2019) 321–342.

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

[6] W. Q. Wang, A reversible data hiding algorithm based on bidirectional difference expansion, Multimedia Tools and Applications 79 (9-10) (2020) 5965–5988.

[7] I. Caciula, H. Coanda, D. Coltuc, Multiple moduli prediction error expansion reversible data hiding, Signal Processing-Image Comnunication 71 (2019) 120–127.

[8] C. Y. Chao, J. C. Lin, Selection of embedding area: A better way to use prediction-error expansion method for reversible hiding, Journal of Information Science and Engineering 36 (3) (2020) 621–641.

[9] Y. Zhou, L. Bao, C. P. Chen, A new 1D chaotic system for image encryption, Signal Processing 97 (2014) 172 – 182.

[10] Z. Hua, Y. Zhou, Image encryption using 2D Logistic-adjusted-Sine map, Information Sciences 339 (2016) 237–253.

[11] R. Lan, Y. Zhou, Z. Liu, X. Luo, Prior knowledge-based probabilistic collaborative representation for visual recognition, IEEE Transactions on Cybernetics 50 (2020) 1498–1508.

[12] R. Lan, L. Sun, Z. Liu, L. Huimin, C. Pang, X. Lu, MADNet: A fast and lightweight network for single image super-resolution, IEEE Transactions on Cybernetics, Accepted, DOI:10.1109/TCYB.2020.2970104.

[13] W. Puech, M. Chaumont, O. Strauss, A reversible data hiding method for encrypted images, Security, Forensics, Steganography, and Watermarking of Multimedia Contents X, Proceedings of SPIE 6819.

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

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

[16] X. Zhang, Z. Qian, G. Feng, Y. Ren, Efficient reversible data hiding in encrypted images, Journal of Visual Communication and Image Representation 25 (2) (2014) 322–328.

[17] J. Zhou, W. Sun, L. Dong, X. Liu, O. C. Au, Y. Y. Tang, Secure reversible image data hiding over encrypted domain via key modulation, IEEE Transactions on Circuits and Systems for Video Technology 26 (3) (2016) 441–452.

[18] Z. Qian, X. Han, X. Zhang, Separable reversible data hiding in encrypted images by n-nary histogram modification, The Third International Conference on Multimedia Technology (2013) 8.

[19] Z. J. Tang, S. J. Xu, D. P. Ye, J. Y. Wang, X. Q. Zhang, C. Q. Yu, Real-time reversible data hiding with shifting block histogram of pixel differences in encrypted image, Journal of Real-Time Image Processing 16 (3) (2019) 709–724.

[20] X. Gui, X. Li, B. Yang, A high capacity reversible data hiding scheme based on generalized prediction-error expansion and adaptive embedding, Signal Processing 98 (0) (2014) 370–380.

[21] K. Ma, W. Zhang, X. Zhao, N. Yu, F. Li, Reversible data hiding in encrypted images by reserving room before encryption, IEEE Transactions on Information Forensics and Security 8 (3) (2013) 553–562.

[22] X. Cao, L. Du, X. Wei, D. Meng, X. Guo, High capacity reversible data hiding in encrypted images by patch-level sparse representation, IEEE Transactions on Cybernetics 46 (5) (2016) 1132–1143.

[23] S. Yi, Y. Zhou, Binary-block embedding for reversible data hiding in encrypted images, Signal Processing 133 (2017) 40–51.

[24] W. Zhang, K. Ma, N. Yu, Reversibility improved data hiding in encrypted images, Signal Processing 94 (0) (2014) 118–127.

[25] D. L. Huang, J. J. Wang, High-capacity reversible data hiding in encrypted image based on specific encryption process, Signal Processing-Image Communication 80, huang, Delu Wang, Jianjun.

[26] M. Li, Y. Li, Histogram shifting in encrypted images with public key cryptosystem for reversible data hiding, Signal Processing 130 (2017) 190–196.

[27] X. Zhang, J. Long, 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 26 (9) (2016) 1622–1631.

[28] C. L. Jiang, Y. L. Pang, Encrypted images-based reversible data hiding in paillier cryptosystem, Multimedia Tools and Applications 79 (1-2) (2020) 693–711.

[29] Y.-C. Chen, T.-H. Hung, S.-H. Hsien, S. C.-W., A new reversible data hiding in encrypted image based on multi-secret sharing and lightweight cryptographic algorithms, IEEE Transactions on Information Forensics and Security 14 (12) (2019) 3332–3343.

[30] X. Wu, J. Weng, W. Yan, Adopting secret sharing for reversible data hiding in encrypted images, Signal Processing 143 (2018) 269–281.

[31] A. Shamir, How to share a secret, Communications of the ACM 22 (11) (1979) 612–613.
[32] P. Li, Z. Q. Liu, C. N. Yang, A construction method of (t, k, n)-essential secret image sharing scheme, Signal Processing-Image Communication 65 (2018) 210–220, li, Peng Liu, Zuquan Yang, Ching-Nung.
[33] L. Bao, S. Yi, Y. Zhou, Combination of sharing matrix and image encryption for lossless (k, n) -secret image sharing, IEEE Transactions on Image Processing 26 (12) (2017) 5618–5631.
[34] K.-Y. CHAO, J.-C. LIN, Secret image sharing: A boolean-operations-based approach combining benefits of polynomial-based and fast approaches, International Journal of Pattern Recognition and Artificial Intelligence 23 (02) (2009) 263–285.
[35] J. Tian, Reversible data embedding using a difference expansion, IEEE Transactions on Circuits and Systems for Video Technology 13 (8) (2003) 890–896. doi:10.1109/tcsvt.2003.815962.