Enabling Large-Capacity Reversible Data Hiding Over Encrypted JPEG Bitstreams

Zhongyun Hua, *Member, IEEE*, Ziyi Wang, Yifeng Zheng, Yongyong Chen, *Member, IEEE*, and Yuanman Li, *Member, IEEE*

***Abstract*— Cloud computing offers advantages in handling the exponential growth of images but also entails privacy concerns on outsourced private images. Reversible data hiding (RDH) over encrypted images has emerged as an effective technique for securely storing and managing confidential images in the cloud. Most existing schemes only work on uncompressed images. However, almost all images are transmitted and stored in compressed formats such as JPEG. Recently, some RDH schemes over encrypted JPEG bitstreams have been developed, but these works have some disadvantages such as a small embedding capacity (particularly for low quality factors), damage to the JPEG format, and file size expansion. In this study, we propose a permutation-based embedding technique that allows the embedding of significantly more data than existing techniques. Using the proposed embedding technique, we further design a large-capacity RDH scheme over encrypted JPEG bitstreams, in which a grouping method is designed to boost the number of embeddable blocks. The designed RDH scheme allows a content owner to encrypt a JPEG bitstream before uploading it to a cloud server. The cloud server can embed additional data (e.g., copyright and identification information) into the encrypted JPEG bitstream for storage, management, or other processing purpose. A receiver can losslessly recover the original JPEG bitstream using a decryption key. Comprehensive evaluation results demonstrate that our proposed design can achieve approximately twice the average embedding capacity compared to the best prior scheme while preserving the file format without file size expansion.**

***Index Terms*— Reversible data hiding, JPEG image, encrypted image, format compatibility, file size preservation.**

Manuscript received 4 March 2022; revised 24 June 2022 and 11 August 2022; accepted 4 September 2022. Date of publication 20 September 2022; date of current version 7 March 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62071142 and Grant 62106063; in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2021A1515011406, Grant 2022A1515010645, Grant 2021A1515110027, and Grant 2022A1515010819; in part by the Shenzhen College Stability Support Plan under Grant GXWD20201230155427003-20200824210638001 and Grant GXWD20201230155427003-20200824113231001; in part by the Foundation for Science and Technology Innovation of Shenzhen under Grant RCBS20210609103708014; and in part by the Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies under Grant 2022B1212010005. This article was recommended by Associate Editor D. Gragnaniello. *(Corresponding author: Yuanman Li.)*

Zhongyun Hua and Yongyong Chen are with the Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies, the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Shenzhen 518055, China (e-mail: huazyum@gmail.com; huazhongyun@hit.edu.cn; yongyongchen.cn@gmail.com).

Ziyi Wang and Yifeng Zheng are with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Shenzhen 518055, China (e-mail: yiwangdow@gmail.com; yifeng.zheng@hit.edu.cn).

Yuanman Li is with the Guangdong Key Laboratory of Intelligent Information Processing, the College of Electronics and Information Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: yuanmanli@szu.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSVT.2022.3208030.

Digital Object Identifier 10.1109/TCSVT.2022.3208030

I. Introduction

MASSIVE images are exponentially produced everyday from different types of imaging devices, such as smartphones and cameras, and the growth is greatly expedited with the rapid development of various Internet of Things applications. Cloud computing is a natural and convenient method for handling exponentially produced images. However, outsourcing image services to the cloud raises critical privacy concerns regarding information-rich images. Reversible data hiding over encrypted images (RDH-EI) is an effective technique for securely storing and managing confidential images in the cloud [1]. This allows an image owner to encrypt an image to protect confidentiality before uploading it to the cloud server. The cloud server can embed additional data into the encrypted image for management, identification, or other processing purposes without accessing the image contents. An authorized receiver can completely decrypt the encrypted image using a decryption key.

According to embedding room vacation strategies, existing RDH-EI schemes can be divided into two classes: reserving room before encryption (RRBE) [2], [3], [4] and vacating room after encryption (VRAE) [5], [6], [7]. RRBE-based schemes require image owners to reserve embedding room in plain images. As these schemes can fully use the pixel redundancy of plain images, they can obtain a large embedding room [3]. However, this strategy requires image owners to reserve the embedding room, which may cause a heavy computational burden because performance-limited terminal devices are typically used in many applications [8]. In contrast, VRAE-based schemes require the cloud server in the encrypted domain to vacate the embedding room [9]. To maintain data redundancy in the encrypted domain for data embedding, most VRAE-based schemes encrypt images using lightweight encryption techniques [10], [11], [12], which exhibit limited strengths in protecting image contents [13], [14]. To achieve a higher security level, some public key encryption schemes are used to encrypt images [15], [16], [17], and data are embedded using the homomorphic properties. However, these public encryption schemes have high computational costs and large data expansions [18].

When images are stored or transmitted over networks, they are first compressed to reduce the storage or communication costs. The JPEG format is a widely used compressed

1051-8215 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

image format because it can achieve a large compression ratio while retaining good visual quality [19]. However, most existing RDH-EI schemes [20], [21], [22] are designed for uncompressed images and cannot be applied to JPEG bitstreams [23]. This is because these RDH-EI schemes embed data using image pixel redundancy, which is eliminated in the JPEG bitstreams. In addition, a JPEG bitstream must maintain a fixed data structure, and data embedding operations in these schemes may destroy the data structure and render the bitstream unreadable by JPEG decoders. This further increases the difficulty of data hiding in JPEG bitstreams [24], [25]. In [26], Qian and Zhang proposed the first RDH method that can embed data into a JPEG bitstream. Subsequently, more studies were devoted to RDH over JPEG bitstreams [27], [28]. However, these methods can not be used directly in encrypted JPEG images. Without considering image distortion, the embedding method of an encrypted JPEG image is different from that of an unencrypted JPEG image. Thus, developing schemes is desirable for RDH over encrypted JPEG bitstreams.

Recently, some RDH schemes over encrypted JPEG bitstreams have been developed [23], [24], [25], [29], [30], [31]. In 2014, Qian *et al*. first proposed an RDH scheme over encrypted JPEG bitstreams [29], in which additional data were encoded with error correction codes to achieve lossless data extraction and image recovery. Later, a new RDH scheme over encrypted JPEG bitstreams with a large embedding capacity was developed [24]. However, this scheme cannot keep the file size unchanged in the marked encrypted JPEG bitstreams. Similar to the scheme in [24], Qian *et al*. developed a scheme in [25] that can embed more data but incurs additional file size expansion. In addition, Sheidani *et al*. [31] proposed an RDH scheme over encrypted JPEG bitstreams using a public-key encryption strategy to encrypt JPEG bitstreams. This scheme can achieve a large embedding capacity without incurring file size changes in the encrypted JPEG bitstreams. However, the JPEG format is destroyed during the encryption and data embedding processes, which may result in encrypted JPEG bitstreams that are unreadable by JPEG decoders [32]. The other schemes in [30] and [23] can achieve both file size and JPEG format preservation during the encryption and data-embedding processes. However, they cannot achieve a large embedding capacity, which is a very important property of cloud-based image services [25]. In summary, existing RDH schemes over encrypted JPEG bitstreams have disadvantages in terms of a small embedding capacity [23], [29], [30], file size changes [24], [25] and JPEG format damage [31].

In this study, we propose a new large-capacity RDH scheme over encrypted JPEG bitstreams, which is a new design to embed data into encrypted domain using a scrambling-embedding technique. Our design accommodates the following workflow. An image owner encrypts a JPEG bitstream and uploads to a cloud server. The cloud server can embed some data, e.g., the copyright and identification information, into the encrypted JPEG bitstream for storage, management, or other processing purposes. A receiver can losslessly recover the original JPEG bitstream using an encryption key. The main contributions of this study are as follows.

- We propose a permutation-based embedding technique on an ordered sequence, which achieves a larger embedding capacity than existing techniques.
- We propose an RDH scheme over encrypted JPEG bitstreams, in which a grouping method is developed to greatly boost the number of embeddable blocks for JPEG bitstreams.
- We conduct a comprehensive evaluation, which shows that our scheme outperforms the best existing scheme in average embedding capacity, while preserving the file format without size expansion.

The remainder of this paper is organized as follows. Section II introduces the JPEG format and previous RDH schemes over JPEG bitstreams. Section III presents the permutation-based embedding technique. Section IV describes the proposed RDH scheme over encrypted JPEG bitstreams. Section V presents the results for the proposed RDH scheme and discusses its properties. Section VI evaluates the proposed scheme and compares it with previous methods. Finally, Section VII concludes the paper.

II. JPEG Format and Related Works

In this section, we introduce the JPEG format, present existing RDH schemes over encrypted JPEG bitstreams, and discuss their properties.

A. JPEG Format

The JPEG format is a widely used image-compression format, and its coding process includes four steps.

1) Blocking: An uncompressed image is first divided into 8×8 equally sized blocks. The color space is converted from the RGB space to the YUV space for a color image, and only the Y channel exists for a grayscale image. In addition, each pixel is subtracted by 128 such that all pixel values can be evenly distributed with a center of 0.

2) DCT: For each block, perform a discrete cosine transform (DCT) to obtain an 8×8 coefficient matrix. The first number in position $(1, 1)$ of the coefficient matrix is called the DC coefficient, and the remaining 63 numbers are called AC coefficients.

3) Quantization: A quantization operation is performed on each coefficient matrix, and a quality factor parameter is used to determine an 8×8 quantization matrix Q. Generally, a lower quality factor indicates greater image data loss and a smaller file size. The quantized coefficient matrix is obtained as $DCT'(x, y) = \text{round}(DCT(x, y)/Q(x, y))$, where DCT is the DCT coefficient matrix and the function $\text{round}(\cdot)$ is used to obtain the nearest integer.

4) Entropy Coding: After quantization, the two-dimensional quantized coefficient matrix is pulled into a one-dimensional array using a zigzag transform. Then the 64 coefficients in each block are encoded into several triples using run length coding [19]. The j-th triple in the i-th block is represented by $p_i^{(j)} = (R_i^{(j)}, S_i^{(j)}, V_i^{(j)})$, where R is the number of continuous zero coefficients, S is the bitstream length of the non-zero coefficient, and V is the bitstream of the non-zero coefficient. Since the DC coefficient is the first number in each block, the

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

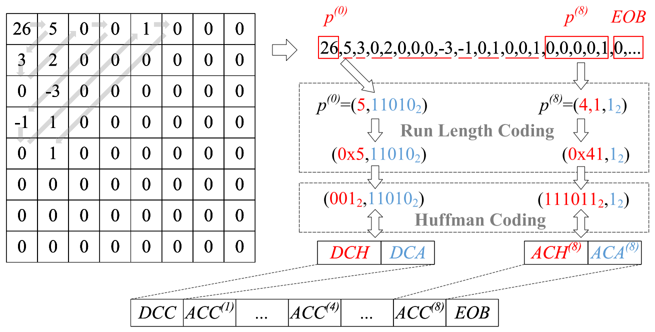

Fig. 1. Numeral example of the entropy encoding result to a quantized coefficient matrix.

R in the triple of the DC coefficient is a constant zero and can be ignored. Then, the first triple of the i-th block can be degenerated into a doublet $p_i^{(0)} = (S_i^{(0)}, V_i^{(0)})$. Consequently, the i-th block $\mathbf{B}_i$ of an image can be represented as $\mathbf{B}_i = \{p_i^{(0)}, p_i^{(1)}, \cdots\}$.

Finally, the i-th block $\mathbf{B}_i$ is encoded as the i-th multiple minimum coded unit $\mathbf{MCU_i} = \{DCC_i, ACC_i^{(1)}, ACC_i^{(2)}, \cdots, EOB\}$. The i-th block's DC code DCC_i includes two parts DCH_i and DCA_i, where DCH_i is the Huffman code of $S_i^{(0)}$ and DCA_i is $V_i^{(0)}$. The i-th block's j-th AC code $ACC_i^{(j)}$ also includes two parts $ACH_i^{(j)}$ and $ACA_i^{(j)}$, where $ACH_i^{(j)}$ is the Huffman code of $R_i^{(j)}$ and $S_i^{(j)}$, and $ACH_i^{(j)}$ is $V_i^{(j)}$. The EOB indicates the end of block. The remaining zeros after the last non-zero AC coefficients do not need to be encoded, because a block has 63 AC coefficients.

To better show the process of entropy coding for a block, we provide a numeral example in Fig. 1. The quantized DCT coefficient matrix is first converted as a 1D array using the zigzag transform. The first value of 26 is the DC coefficient, and the remaining 63 values are the AC coefficients. Using run-length coding, the DC coefficient is encoded as a doublet $p^{(0)} = (5, 11010_2)$ and the AC coefficients are encoded as triples, for example, $p^{(8)} = (4, 1, 1_2)$. Then, DCH and $ACH^{(j)}$ can be obtained by encoding S_0 in $p^{(0)}$ and $(R^{(j)}, S^{(j)})$ in $p^{(j)}$ using the Huffman code. For example, $S_0 = 5$ is encoded as $DCH = 001_2$ and $(R^{(8)}, S^{(8)}) = (4, 1)$ is encoded as $ACH = 111011_2$.

B. RDH-EI Schemes Over Encrypted JPEG Bitstreams

Recently, some RDH-EI schemes over encrypted JPEG bitstreams have been developed with different properties. Depending on whether the file size and format can be preserved, these schemes can be discussed as follows.

1) Without File Size Preservation: This type of schemes can achieve a large embedding capacity; however, the size of the marked encrypted JPEG bitstreams differs from that of the original JPEG bitstreams. The authors of [24] proposed such a scheme by dividing an original JPEG bitstream into two parts by blocks. The first part includes randomly selected DCT coefficients that are encrypted by a secret key with an XOR operation. The other part of data is placed into the header of the JPEG bitstream after rearranging and encrypting. The embedding capacity is vacated by compressing the second part of data. The embedded data were extracted using the high pixel redundancy of adjacent blocks. In addition, image recovery and data extraction are simultaneously performed. However, some practical applications require image recovery and data extraction to be separable. For example, in the cloud storage, additional data are extracted by the cloud server for storage, whereas the image is recovered by an authorized receiver.

Similar to the scheme of [24], the authors in [25] divided a JPEG bitstream into two parts by blocks, in which one part constructs a new JPEG bitstream, and the other part is encrypted and hidden in the JPEG header. Additional data are embedded into the new JPEG bitstream using variable-length coding and histogram shifting. It can achieve a larger embedding capacity than the scheme in [24], and its data extraction and image recovery are separable. However, this scheme causes an increase in the file size in the encrypted bitstreams.

2) Without Format Preservation: The schemes in [24] and [25] cannot preserve the JPEG format, because the JPEG header changes during encryption and data embedding.

The authors of [31] proposed a separable RDH scheme over encrypted JPEG bitstreams. This scheme uses an asymmetric encryption strategy to encrypt JPEG bitstreams. The two most significant bits of each DCH are used for data embedding and the image is recovered using the characteristics of the Huffman code in the JPEG format. It achieves good embedding capacity and has a high security level for chosen-plaintext attack. However, this scheme destroys the JPEG format during encryption and embedding processes. A JPEG bitstream is unreadable by JPEG decoders, and thus, causes problems for file compatibility if its JPEG format is destroyed [32].

3) With Both File Size and Format Preservation: File size preservation indicates that the encryption and data embedding processes do not cause file size changes in the marked encrypted JPEG bitstreams. However, according to the discussions in [19] and [32], minor changes in the final file size may occur when processing JPEG bitstreams because of the byte alignment in the encoding procedure, which is unavoidable. Thus, the minor file size change caused by the byte alignment is ignored in all RDH schemes over encrypted JPEG bitstreams [23], [29], [30].

In 2014, Qian *et al.* proposed RDH over encrypted JPEG bitstreams with both file size and format preservation [29]. ACAs and DCAs are encrypted by the stream cipher, whereas ACHs and DCHs remain unchanged. Additional data are embedded by flipping the least significant bits of the encrypted ACAs of each embeddable block, and the image is recovered using the blocking artifact function. As an early study, the embedding capacity is not high and the image could not be recovered losslessly before data extraction. In addition, the encrypted DCAs may cause an overflow of the quantized DC coefficients because of differential pulse code modulation.

The scheme in [30] losslessly compresses part of the JPEG bitstreams for data embedding before encryption. The image is encrypted by performing an XOR operation on the DCAs and ACAs, and the additional data are embedded into the room preserved by compression. Because data extraction and image recovery are independent, this scheme is separable. The authors of [23] proposed a rotation-based scheme, in which a rotation-embedding technique was developed for data embedding by rotating the RSV triples, and an encryption strategy introduced in [32] was used for image encryption.

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

This method can preserve both the file size and JPEG format during data embedding and image encryption, as well as avoid data overflow. For all these schemes with both file size and format preservation, their embedding capacities are not high, particularly with low quality factors. However, in some practical applications such as cloud storage, a cloud server may be willing to embed more data into encrypted bitstreams for further processing, such as image retrieval [33]. Thus, designing RDH schemes is meaningful for encrypted JPEG bitstreams with high security and a large embedding capacity.

III. New Data Embedding Technique Based on Permutation

In this section, we introduce a new data embedding technique based on permutation, develop a fast search implementation to it, and finally discuss its properties.

A. Permutation-Based Embedding

Here, we present a new permutation-based embedding technique for an ordered sequence. The main concept is that we first calculate the number of different permutations of the sequence and then provide a unique number for each permutation. Finally, we treat the data to be embedded as a distance and change the original ordered sequence to a permuted sequence according to the distance.

First, we introduce the concept of a multiset. Suppose that $\mathbf{S} = \{a_1 \cdot b_1, \cdots, a_v \cdot b_v \cdots, a_K \cdot b_K\}$ is a descending multiset with N elements, where a_v indicates the number of b_v. For example, a multiset $\mathbf{S} = \{2 \cdot 4, 1 \cdot 2, 2 \cdot 1\}$ can be expressed as $\mathbf{S} = \{4, 4, 2, 1, 1\}$, which contains three different elements. For the multiset $\mathbf{S} = \{a_1 \cdot b_1, \cdots, a_v \cdot b_v \cdots, a_K \cdot b_K\}$ with N elements, the number of permutations can be calculated as

$$|\mathbf{S}| = \frac{N!}{\prod_{v=1}^{K}(a_v!)}. \tag{1}$$

Then, we number these $|\mathbf{S}|$ permutations by comparing their elements from left to right. Thus, the sequence $\{a_1 \cdot b_1, \cdots, a_v \cdot b_v \cdots, a_K \cdot b_K\}$ is numbered 0, whereas the sequence $\{a_K \cdot b_K, \cdots, a_v \cdot b_v \cdots, a_1 \cdot b_1\}$ is numbered as $|\mathbf{S}| - 1$. For example, for the descending multiset $\mathbf{S} = \{2 \cdot 4, 1 \cdot 2, 2 \cdot 1\}$, there are 30 different permutations according to Eq. (1). Then the sequence $\{4, 4, 2, 1, 1\}$ is numbered 0, the sequence $\{4, 4, 1, 2, 1\}$ is numbered 1, and the sequence $\{1, 1, 2, 4, 4\}$ is numbered 29.

Because each permutation corresponds to a value, the largest bit number that can be embedded into the descending multiset $\mathbf{S}$ can be calculated as:

$$C = \lfloor \log_2 |\mathbf{S}| \rfloor = \left\lfloor \log_2 \frac{N!}{\prod_{v=1}^{K}(a_v!)} \right\rfloor. \tag{2}$$

Thus, when embedding C bits into the descending multiset $\mathbf{S}$, we first convert the C bits to a decimal value and then change the descending multiset as the sequence that corresponds to this value.

The entire data embedding process for the descending multiset $\mathbf{S} = \{a_1 \cdot b_1, \cdots, a_v \cdot b_v \cdots, a_K \cdot b_K\}$ with N elements can be described as follows.

- *Step 1*: Calculate the permutation number of $|\mathbf{S}|$ according to Eq. (1);
- *Step 2*: Calculate the embedding capacity C according to Eq. (2);
- *Step 3*: For the C bits $\mathbf{D} = \{d_1, d_2, \cdots, d_C\}$ to be embedded, convert them to a decimal integer D_I.
- *Step 4*: Number these $|\mathbf{S}|$ different permutations of $\mathbf{S}$ by comparing their elements from left to right;
- *Step 5*: Embed D_I into $\mathbf{S}$ by selecting the sequence $\mathbf{S}'$ numbered as D_I as the output sequence.

B. Fast Search Implementation

The permutation operation of the multiset is complex, with a heavy computational cost. To address this problem, we designed a fast search implementation to determine the permuted sequence $\mathbf{S}'$ corresponding to the value of D_I. For a descending multiset $\mathbf{S} = \{a_1 \cdot b_1, \cdots, a_v \cdot b_v \cdots, a_K \cdot b_K\}$ with N elements, we first calculate the number of different permutations beginning with b_v, that is, the number of permutations in multiset $\mathbf{S}/\{b_v\} = \{a_1 \cdot b_1, \cdots, (a_v - 1) \cdot b_v \cdots, a_K \cdot b_K\}$:

$$t_v = \frac{(N-1)!}{a_1! \cdots (a_v - 1)! \cdots a_K!}. \tag{3}$$

Evidently $|\mathbf{S}| = t_1 + \cdots + t_K$. Letting $t_0 = 0$, we can calculate the permutation range beginning with b_u as $R_u = [\sum_{v=0}^{v=u-1} t_v, \sum_{v=0}^{v=u} t_v)$. Then, if $D_I \in R_u$, we can determine that the first element of the search sequence is b_u. After determining the first element b_u, we exclude a b_u from the sequence and update the value D_I by $D_I = D_I - \sum_{v=0}^{v=u-1} t_v$. Repeat this to determine all elements in the searched sequence. The entire fast search implementation can be described as follows.

- *Step 1*: Initialize an empty sequence $\mathbf{S}'$;
- *Step 2*: If $D_I = 0$, append the remaining elements in $\mathbf{S}$ to $\mathbf{S}'$, that is, $\mathbf{S}' = \mathbf{S}' \cup \mathbf{S}$. Then the operation ends and we find the sequence $\mathbf{S}'$ that corresponds to D_I;
- *Step 3*: Count the number of elements with different values in $\mathbf{S}$ as K. Calculate the number of different permutations beginning with b_v according to Eq. (3) to obtain a K-length vector $\mathbf{T} = \{t_1, t_2, \cdots, t_K\}$.
- *Step 4:* Calculate the permutation range beginning with $b_1, \cdots, b_K$ to obtain $R_1, \cdots, R_K$.
- *Step 5:* Determine the first element of the searched sequence as b_u if $D_I \in R_u$.
- *Step 6:* Remove a b_u from $\mathbf{S}$ and append it to the $\mathbf{S}'$. Update D_I by subtracting the lower bound of R_u.
- *Step 7*: Repeat *Steps 2* to *6*.

Algorithm 1 presents the pseudo-code of the data embedding procedures using the fast search implementation, while Algorithm 2 shows the pseudo-code of the data extraction procedures. The fast search implementation can reduce the time complexity of data embedding from $O(N!)$ to $O(N^2)$, which can significantly improve the data embedding efficiency. For the original algorithm, the computational complexity is $O(n!)$ to search for an n-length vector with n different elements. For the fast search algorithm, we must only traverse each element and calculate the number of sequences with different elements. Thus, the complexity in the worst case is only $O(n^2)$.

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

To better illustrate the data embedding and extraction procedures, a numeral example is provided in Fig. 2. Suppose that the descending multiset is $\mathbf{S} = \{4, 4, 3, 2, 1\}$. According to Eqs. (1) and (2), the embedding capacity $C = 5$ and embedded bits are assumed to be $\mathbf{D} = \{0, 1, 1, 0, 0\}$. The decimal number of $\mathbf{D}$ is obtained as $D_I = 12$. In the embedding phase, first calculate the initial $\mathbf{T} = \{24, 12, 12, 12\}$. Then, the ranges of the permutations with the first numbers "4," "3," "2,"and "1" are $[0, 24)$, $[24, 36)$, $[36, 48)$ and $[48, 60)$, respectively. As $D_I \in [0, 24)$, we determine the first number to be "4." Then, update $\mathbf{S} = \mathbf{S}\backslash\{4\} = \{4, 3, 2, 1\}$, $\mathbf{S}' = \mathbf{S}'\cup\{4\} = \{4\}$ and $D_I = 12-0 = 12$. Second, calculate $T = \{6, 6, 6, 6\}$, and the ranges of the permutations with the first numbers "4," "3," "2," and "1" are $[0, 6)$, $[6, 12)$, $[12, 18)$ and $[18, 24)$, respectively. As $D_I \in [12, 18)$, we determine the second number to be "2." Then, update $\mathbf{S} = \mathbf{S} \setminus \{2\} = \{4, 3, 1\}$, $\mathbf{S}' = \mathbf{S}' \cup \{2\} = \{4, 2\}$ and $D_I = 12 - 12 = 0$. Third, as $D_I = 0$, we can get the result $\mathbf{S}' = \mathbf{S}' \cup \mathbf{S} = \{4, 2, 4, 3, 1\}$ and the embedding process ends.

Algorithm 1 Data Embedding Procedures

```
Input: Binary bits $\mathbf{D} = \{d_1, d_2, \cdots, d_C\}$ and the descending
    multiset $\mathbf{S} = \{a_1 \cdot b_1, \cdots, a_v \cdot b_v \cdots, a_K \cdot b_K\}$ with $N$
    elements;
Output: Permuted sequence $\mathbf{S}'$.
 1: $D_I \leftarrow \text{bin2dec}(\mathbf{D})$;
 2: while $D_I \neq 0$ do
 3:   $K \leftarrow \text{getK}(\mathbf{S})$;
 4:   $\mathbf{T} = \{t_1, t_2, \cdots, t_v, \cdots, t_K\}$;
 5:   for $v \leftarrow 1 : K$ do
 6:     $t_v \leftarrow |\mathbf{S} \setminus \{b_v\}|$;
 7:   end for
 8:   $u \leftarrow 1$;
 9:   while $D_I \geq t_u$ do
10:     $D_I \leftarrow D_I - t_u$, $u \leftarrow u + 1$;
11:   end while
12:   $\mathbf{S}' \leftarrow \mathbf{S}' \cup \{b_u\}$, $\mathbf{S} \leftarrow \mathbf{S} \setminus \{b_u\}$ ;
13: end while
14: return $\mathbf{S}' \leftarrow \mathbf{S}' \cup \mathbf{S}$;
```

In the extraction phase, we obtain $\mathbf{S} = \{4, 4, 3, 2, 1\}$ by sorting $\mathbf{S}' = \{4, 2, 4, 3, 1\}$ in descending order. The data extraction is similar to the data embedding. First, initialize $D_I = 0$, calculate $\mathbf{T} = \{24, 12, 12, 12\}$ and the ranges of the permutations with the first numbers "4," "3," "2," and "1" are $[0, 24)$, $[24, 36)$, $[36, 48)$ and $[48, 60)$, respectively. As the first number of $\mathbf{S}'$ is $s_1 = b_1 = 4$, $D_I = 0 + 0 = 0$, and $\mathbf{S} = \mathbf{S}\backslash\{b_1\} = \{4, 3, 2, 1\}$, $\mathbf{S}' = \mathbf{S}'\backslash\{s_1\} = \{2, 4, 3, 1\}$. Second, calculate $T = \{6, 6, 6, 6\}$ and the ranges of the permutations with the first numbers "4," "3," "2," and "1" are $[0, 6)$, $[6, 12)$, $[12, 18)$, and $[18, 24)$, respectively. As the first number of $\mathbf{S}'$ is $s_1 = b_3 = 2$, $D_I = 0 + 12 = 12$ and $\mathbf{S} = \mathbf{S} \setminus \{b_3\} = \{4, 3, 1\}$, $\mathbf{S}' = \mathbf{S}' \setminus \{s_1\} = \{4, 3, 1\}$. Third, as $\mathbf{S} = \mathbf{S}'$, the data extraction ends and the embedded data value is $D_I = 12$. From Eq. (2), the embedding capacity $C = 5$. Subsequently, D_I is converted into five bits $\mathbf{D} = \{0, 1, 1, 0, 0\}$.

Algorithm 2 Data Extraction Procedures

```
Input: Permuted sequence $\mathbf{S}' = \{s_1, s_2 \cdots, s_N\}$;
Output: Binary bits $\mathbf{D} = \{d_1, d_2, \cdots, d_C\}$.
 1: $\mathbf{S} = \{a_1 \cdot b_1, \cdots, a_v \cdot b_v \cdots, a_K \cdot b_K\}$;
 2: $C \leftarrow |\mathbf{S}|$, $D_I \leftarrow 0$;
 3: while $\mathbf{S}' \neq \mathbf{S}$ do
 4:   $K \leftarrow \text{getK}(\mathbf{S})$;
 5:   $\mathbf{T} = \{t_1, t_2, \cdots, t_v, \cdots, t_K\}$;
 6:   for $v \leftarrow 1 : K$ do
 7:     $t_v \leftarrow |\mathbf{S} \setminus \{b_v\}|$;
 8:   end for
 9:   $s_1 \leftarrow \text{fisrtNumber}(\mathbf{S}')$, $u \leftarrow 1$;
10:   while $s_1 < b_u$ do
11:     $D_I \leftarrow D_I + t_u$, $u \leftarrow u + 1$;
12:   end while
13:   $\mathbf{S} \leftarrow \mathbf{S} \setminus \{b_u\}$, $\mathbf{S}' \leftarrow \mathbf{S}' \setminus \{s_1\}$;
14: end while
15: return $\mathbf{D} \leftarrow \text{dec2bin}(D_I, C)$;
```

C. Embedding Capacity Discussion

Using the permuted sequence to present the embedded data, the proposed permutation-based embedding technique can achieve a high embedding capacity. It can also achieve relatively high efficiency using a fast search implementation. Here, we investigate its embedding capacity against different types of descending multisets and compare its capacity with that of similar techniques introduced in [34], [35], and [23]. Note that the AC-coefficient-based embedding technique in [34] is used as the competing method because it follows the same principle as ours.

These techniques and ours embed data by changing the order of an ordered sequence, and they are scrambling-embedding techniques. The data embedding techniques in [34], [35], and [23] embed data over an ordered sequence by rotating it. The techniques in [34] and [35] can rotate the sequence in both directions, whereas those in [23] can only rotate the sequence in one direction. Our technique first numbers each permutation of the ordered sequence, and then treats the data to be embedded as a distance. The embedding process involves determining the permuted sequence that corresponds to the distance. For an ordered multiset $\mathbf{S} = \{a_1 \cdot b_1, \cdots, a_v \cdot b_v \cdots, a_K \cdot b_K\}$ with N elements, the embedding capacity of the technique in [34] and [35] can be calculated as

$$C_1 = \lfloor \log_2 2N \rfloor = \lfloor \log_2 N \rfloor + 1, \tag{4}$$

and the embedding capacity of the technique in [23] is calculated as

$$C_2 = \lfloor \log_2 N \rfloor. \tag{5}$$

Note that K should satisfy $K \geq 3$ in [34] and [35] and $K \geq 2$ in [23]. Accorded to Eq. (2), when $K = 1$, the embedding capacity of the proposed permutation-based embedding technique is $C = 0$. Thus, K should satisfy $K \geq 2$ in the proposed technique.

Lemma 1: For an ordered multiset $\mathbf{S} = \{a_1 \cdot b_1, \cdots, a_v \cdot b_v \cdots, a_K \cdot b_K\}$ *with* N *elements and* $K \geq 2$*, the embedding*

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

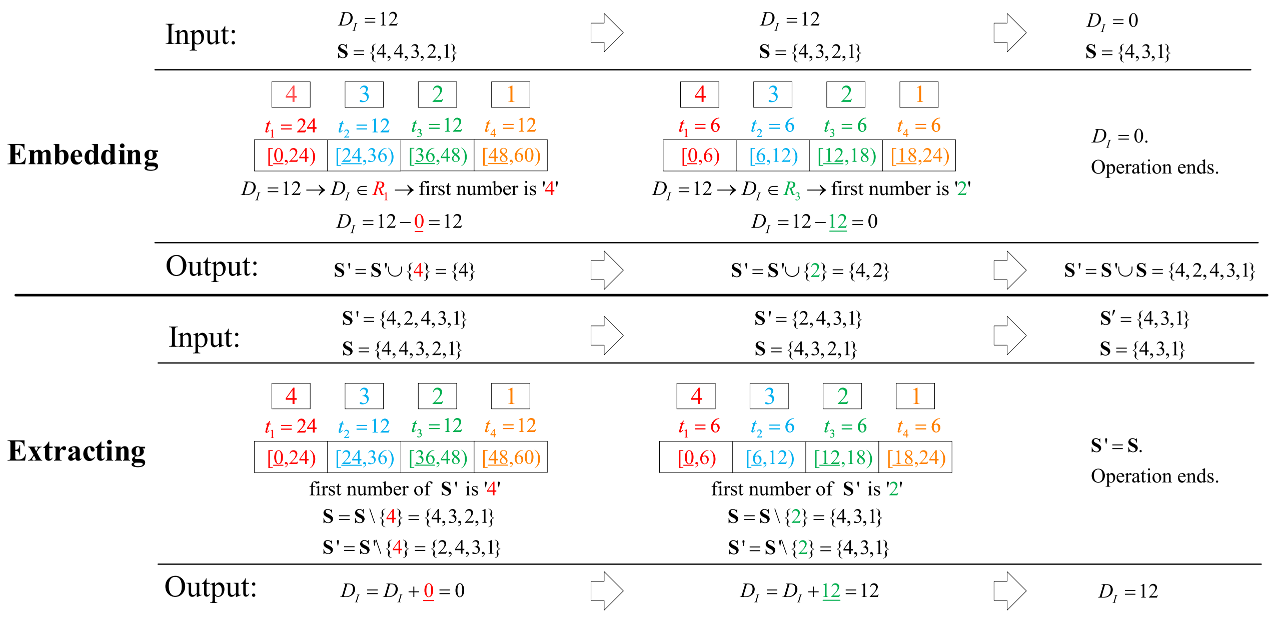

Fig. 2. Numeral example of data embedding and extraction with the multiset $\mathbf{S} = \{2 \cdot 4, 1 \cdot 3, 1 \cdot 2, 1 \cdot 1\}$ and embedded data $\mathbf{D} = \{0, 1, 1, 0, 0\}$.

capacity of our technique is given by Eq. (2), which is not lower than the embedding capacity shown in Eqs. (4) and (5).

Proof: When $K = 2$, $(a_1!)(a_2!)$ can evidently achieve the maximum value $(N-1)!$ when one of a_1 and a_2 is 1 and the other is $N-1$. Thus $(a_1!)(a_2!) \le (N-1)!$ and the embedding capacity of the proposed scheme in Eq. (2) can be rewritten as follows:

$$\begin{aligned} C &= \left\lfloor \log_2 \frac{N!}{(a_1!)(a_2!)} \right\rfloor \\ &\ge \left\lfloor \log_2 \frac{N!}{(N-1)!} \right\rfloor \\ &= \lfloor \log_2 N \rfloor = C_2. \end{aligned} \tag{6}$$

This is unavailable for the techniques in [34] and [35] when $K = 2$.

When $K \ge 3$, $(a_1!)(a_2!) \cdots (a_K!)$ can evidently achieve the maximum value $(N-K+1)!$ when one of a_v is $N-K+1$ and the others are one. Thus $(a_1!)(a_2!) \cdots (a_K!) \le (N-K+1)!$ and the embedding capacity of the proposed technique in Eq. (2) can be rewritten as follows:

$$\begin{aligned} C &= \left\lfloor \log_2 \frac{N!}{(a_1!)(a_2!) \cdots (a_K!)} \right\rfloor \\ &\ge \left\lfloor \log_2 \frac{N!}{(N-K+1)!} \right\rfloor \\ &= \lfloor \log_2 N(N-1) \cdots (N-K+2) \rfloor. \end{aligned} \tag{7}$$

Because $K \ge 3$, we can obtain $C \ge \lfloor \log_2 N(N-1) \rfloor \ge C_1 > C_2$. This completes this proof. ■

Table I lists the embedding capacities of the different techniques over a decreasing multiset $\mathbf{S}$ of length 5. When $K = 1$, the embedding capacities of all techniques are zero, because the five elements in $\mathbf{S}$ are the same. When $K = 2$, the embedding capacity of [34], [35] is still zero, because the technique in [34] and [35] embed data by circularly shifting and flipping the original sequence, and same permuted sequences may be obtained when embedding different values under this case. This causes that the embedding values cannot be extracted correctly. However, the embedding capacity of [23] and the proposed scheme is 2. Six different types of sequences exist when $K = 3$. The embedding capacity of [34], [35] and our technique is 3, whereas that of [34], [35] is 2. When $K > 4$, the proposed technique can achieve a larger embedding capacity than the other techniques. Thus, the proposed permutation-based embedding technique can achieve a large embedding capacity.

TABLE I
EMBEDDING CAPACITY OF DIFFERENT TECHNIQUES OVER A DECREASING MULTISET OF LENGTH 5

Numbers	Sequence	Refs. [34], [35]	Ref. [23]	Ours
$K = 1$	$\{b_1, b_1, b_1, b_1, b_1\}$	0	0	0
$K = 2$	$\{b_2, b_1, b_1, b_1, b_1\}$	0	2	2
	$\{b_2, b_2, b_1, b_1, b_1\}$	0	2	2
	$\{b_2, b_2, b_2, b_1, b_1\}$	0	2	2
	$\{b_2, b_2, b_2, b_2, b_1\}$	0	2	2
$K = 3$	$\{b_3, b_3, b_3, b_2, b_1\}$	3	2	3
	$\{b_3, b_3, b_2, b_2, b_1\}$	3	2	3
	$\{b_3, b_2, b_2, b_2, b_1\}$	3	2	3
	$\{b_3, b_3, b_2, b_1, b_1\}$	3	2	3
	$\{b_3, b_2, b_2, b_1, b_1\}$	3	2	3
	$\{b_3, b_2, b_1, b_1, b_1\}$	3	2	3
$K = 4$	$\{b_4, b_4, b_3, b_2, b_1\}$	3	2	4
	$\{b_4, b_3, b_3, b_2, b_1\}$	3	2	4
	$\{b_4, b_3, b_2, b_2, b_1\}$	3	2	4
	$\{b_4, b_3, b_2, b_1, b_1\}$	3	2	4
$K = 5$	$\{b_5, b_4, b_3, b_2, b_1\}$	3	2	5

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

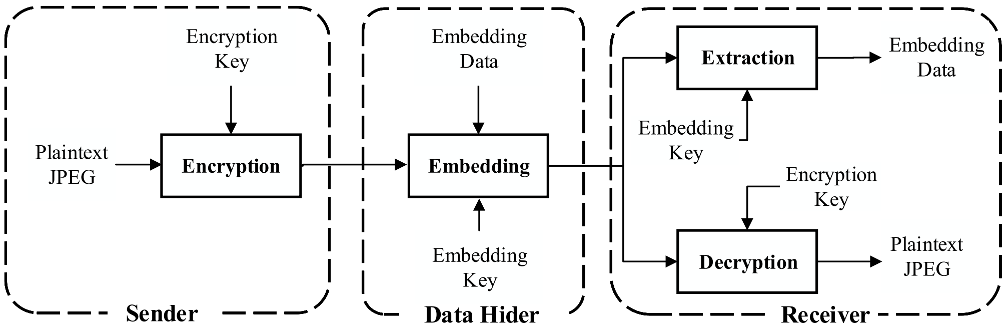

Fig. 3. Framework of our large-capacity RDH scheme over encrypted JPEG bitstreams.

IV. Large-Capacity RDH Scheme Over Encrypted JPEG Bitstreams

In this section, we introduce a large-capacity RDH scheme over encrypted JPEG bitstreams, and Fig. 3 illustrates this framework.

A. JPEG Encryption

The JPEG bitstream consists of a structured data in a special format. Its structure is damaged if it directly encrypted using existing encryption methods such as the Advanced Encryption Standard (AES), and then becomes unreadable for JPEG decoders. As a result, the JPEG bitstreams should be encrypted using specific strategies.

According to the JPEG format introduced in Section II-A, an entire JPEG bitstream can be divided into AC and DC coefficients. The DC coefficients include DCHs and DCAs, whereas the AC coefficients include ACHs and ACAs. ACHs and DCHs are the Huffman codes of lengths and their values should remain unchanged. Otherwise, the JPEG bitstream may become unreadable.

When encrypting the AC coefficients, we encrypt the ACAs using AES encryption, which has a high security level. First, we extract all ACAs from a JPEG bitstream to obtain a bitstream,

$$\mathbf{B}_{ACA} = ACA_1^{(1)}||ACA_1^{(2)}||\cdots||ACA_i^{(1)}||ACA_i^{(2)}||\cdots \quad (8)$$

Then, we encrypt the $\mathbf{B}_{ACA}$ using AES with an encryption key.

Because the values of the ACHs cannot be changed, we randomly scramble their positions among the blocks. The ACHs of each block consist of the Rs and Ss. Then, this scrambling can not change the positions of the Rs and Ss within each block. For example, the AC coefficients of i-th block can be decoded as

$$\mathbf{B}_i^{AC} = \{(R_i^{(1)}, S_i^{(1)}, V_i^{(1)}), \cdots, (R_i^{(n)}, S_i^{(n)}, V_i^{(n)})\}.$$

The ACH of the i_1-th and i_2-th blocks can be decoded as

$$\mathbf{B}_{i_1}^{ACH} = \{(R_{i_1}^{(1)}, S_{i_1}^{(1)}), (R_{i_1}^{(2)}, S_{i_1}^{(2)}), \cdots, (R_{i_1}^{(n_1)}, S_{i_1}^{(n_1)})\},$$

and

$$\mathbf{B}_{i_2}^{ACH} = \{(R_{i_2}^{(1)}, S_{i_2}^{(1)}), (R_{i_2}^{(2)}, S_{i_2}^{(2)}), \cdots, (R_{i_2}^{(n_2)}, S_{i_2}^{(n_2)})\},$$

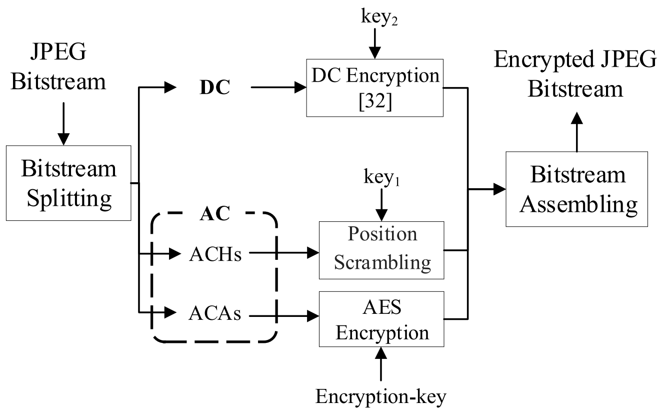

Fig. 4. JPEG encryption scheme.

respectively. When exchanging the positions of $\mathbf{B}_{i_1}^{ACH}$ and $\mathbf{B}_{i_2}^{ACH}$, the Rs and Ss within each block remain unchanged.

To achieve better security, the scrambling key Key_1 is obtained from the $\mathbf{B}_{ACA}$ using a SHA-512 hash algorithm.

For the DC coefficients, the DCAs are not the real values of the DC coefficients because differential pulse modulation coding is used. Thus, directly encrypting the DCAs may cause overflow in the JPEG decoding process. The encryption method introduced in [32] is used to encrypt the DC coefficients. It has a high security level with both file size and format preservation, and can avoid overflow [32]. To achieve better security, the encryption key Key_2 is also obtained from $\mathbf{B}_{ACA}$ using a SHA-512 hash algorithm.

After encryption, the encrypted data must be reassembled. In this process, the encrypted DCHs and ACHs are encoded using a Huffman table. Fig. 4 shows the processes of encrypting the AC and DC coefficients. The decryption processes are the opposite operations of the encryption processes and can be described as follows.

- *Step 1*: Split the encrypted JPEG bitstream to construct the $\mathbf{B}'_{ACA}$;
- *Step 2*: Decrypt the AC bitstream $\mathbf{B}'_{ACA}$ using the encryption key and obtain the original $\mathbf{B}_{ACA}$;
- *Step 3*: Calculate the encryption key for decrypting ACHs $Key_1 = Hash(\hat{\mathbf{B}}_{ACA})$; Calculate the encryption key for decrypting the DC coefficients $Key_2 = Hash(\tilde{\mathbf{B}}_{ACA})$;

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

- *Step 4*: Recover the DC coefficients using Key_2.
- *Step 5*: Recover the ACHs using Key_1.
- *Step 6*: Assemble the original JPEG bitstream.

After encryption, the encrypted JPEG bitstream is sent to the data hider such as a cloud sever.

B. Data Embedding

When a data hider receives the encrypted JPEG bitstream, he/she can embed additional data, e.g., authorization and copyright information, to the encrypted bitstream for storage, management or other processes.

Here, we use the proposed permutation-based embedding technique to embed additional data. As discussed in Section III, the permutation-based embedding technique operates over an ordered sequence. Thus, we use the AC coefficients of each block to embed data.

1) Decision of Embeddable Blocks: First, we determine the embeddable blocks. As discussed in Section II-A, the AC coefficients in each block are encoded into several triples $p_i^{(j)} = (R_i^{(j)}, S_i^{(j)}, V_i^{(j)})$. In a quantized coefficient matrix, the upper-left values are typically larger and have fewer zeros than bottom-right values. Because R represents the number of continuous zero coefficients, and S represents the bitstream length of the non-zero coefficient, R shows an ascending order and S shows a descending order after the zigzag-transform. Then we can use their orders to construct an ordered sequence.

Note that not all blocks have strict orders for the Rs and Ss of AC triples. Thus, we should first determine the embeddable blocks. To increase the number of embeddable blocks, we uniformly divide the AC triples in each block into several groups and calculate a number called 'label number' for each group. Suppose that we divide the N_{ac} AC triples $(R^{(j)}, S^{(j)}, V^{(j)})$ of a block into M groups, and each group has $\lfloor N_{ac}/M \rfloor$ AC triples. For the $\lfloor N_{ac}/M \rfloor$ AC triples of the h-th $(1 \le h \le M)$ group, we calculate its label number as

$$L^{(h)} = \sum_{l=1}^{\lfloor N_{ac}/M \rfloor} \frac{(S^{(l)})^2}{R^{(l)} + 1}. \tag{9}$$

When $L^{(h)} \ge L^{(h+1)}$ and $L^{(1)} > L^{(M)}$ are satisfied, the M groups are in descending order and the block is embeddable. When the descending order is not satisfied, the block is a failed block. In addition, if the number of AC triples N_{ac} is less than M, the block is useless.

Our grouping strategy differs from that in [34]. For the grouping method in [34], the number of AC triples in each group was fixed at two, and the group numbers of the different blocks can be different. However, the settings of our method are the opposite. In particular, for different blocks, the group numbers are the same, whereas the number of the AC triples in each group can be different. The sequence size of each block can be reduced by dividing the AC triples into groups. However, the number of embeddable blocks is significantly increased, because this strategy can greatly strengthen the order of the Rs and Ss in the AC coefficients. The embedding capacity is evidently determined by the group number M and we find that most natural images can achieve the largest embedding capacity when $M = 4$, which is discussed in Section V-A.

TABLE II

EMBEDDING SPACE WASTE OF THE MULTISET $\mathbf{S}_L$ WITH DIFFERENT K WHEN $M = 4$. THE $|\mathbf{S}_L|$ IS THE NUMBER OF PERMUTATION OF $\mathbf{S}_L$, THE C IS THE EMBEDDING CAPACITY, AND THE WASTED EMBEDDING SPACE IS $W = |\mathbf{S}_L| - 2^C$

K	$\mathbf{S}_L$	$\lvert\mathbf{S}_L\rvert$	C	W
$K=2$	$\{b_2, b_1, b_1, b_1\}$	4	2	0
	$\{b_2, b_2, b_1, b_1\}$	6	2	2
	$\{b_2, b_2, b_2, b_1\}$	4	2	0
$K=3$	$\{b_3, b_3, b_2, b_1\}$	12	3	4
	$\{b_3, b_2, b_2, b_1\}$	12	3	4
	$\{b_3, b_2, b_1, b_1\}$	12	3	4
$K=4$	$\{b_4, b_3, b_2, b_1\}$	24	4	8

2) Embedding Strategy: For each embeddable block with M groups of AC triples, we can calculate M label numbers $L^{(1)}, L^{(2)}, \cdots, L^{(M)}$ with descending order. Then a multiset in descending order is constructed as $\mathbf{S}_L = \{L^{(1)}, L^{(2)}, \cdots, L^{(M)}\}$. We can embed additional data into $\mathbf{S}_L$ using the permutation-based embedding technique. According to Eq. (2), the embedding capacity of the υ-th embeddable block can be calculated as

$$C_\upsilon = \lfloor \log_2 |\mathbf{S}_L^{(\upsilon)}| \rfloor, \tag{10}$$

where $|\mathbf{S}_L^{(i)}|$ is the number of permutations.

Not all permutations can be utilized to embed data. Suppose the grouping number is set to $M = 4$. The block is an embeddable block only when the multiset has a descending order, which means that $K \ge 2$. In addition, because $K \le M$, we obtain $K \in \{2, 3, 4\}$. Then for the multiset $\mathbf{S}_L$ with different Ks, we can calculate the number of permutations $|\mathbf{S}_L|$ using Eq. (2) and embedding capacity C according to Eq. (10). Table II shows the wasted embedding space. Evidently, some embedding space is wasted. For example, for multiset $\mathbf{S}_L = \{b_3, b_2, b_1, b_1\}$, the number of permutations is $|\mathbf{S}_L| = \frac{4!}{1!1!2!} = 12$, and its embedding capacity is $C = \lfloor \log_2 |\mathbf{S}_L| \rfloor = 3$. Thus, the wasted embedding space is $W = |\mathbf{S}_L| - 2^C = 4$.

By combining multiple blocks for embedding, the amount of waste can be reduced. Suppose that P embeddable blocks are combined for data embedding and their multisets of label numbers are $\mathbf{S}_L^{(1)}, \mathbf{S}_L^{(2)}, \cdots, \mathbf{S}_L^{(P)}$. We can calculate the combined embedding capacity C' as

$$\begin{aligned} C' &= \lfloor \log_2 (\prod_{\upsilon=1}^{P} |\mathbf{S}_L^{(\upsilon)}|) \rfloor = \lfloor \sum_{\upsilon=1}^{P} \log_2 |\mathbf{S}_L^{(\upsilon)}| \rfloor \\ &\ge \sum_{\upsilon=1}^{P} \lfloor \log_2 |\mathbf{S}_L^{(\upsilon)}| \rfloor = C_1 + C_2 + \cdots + C_P. \end{aligned} \tag{11}$$

Thus, the embedding capacity can be enhanced by combining several embeddable blocks.

With P combined blocks, we first obtain the embedding capacity C' from Eq. (11) and then obtain the embedded value D by transforming the C' bits into a decimal

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

value. Finally, we divide the value D into P parts, that is $D_I^{(1)}, D_I^{(2)}, \cdots, D_I^{(P)}$, and separately embed them into the P blocks. The idea is to ordinally embed D into the P multisets of the P blocks. In particular, we embed the value into the multisets of label numbers from the first $\mathbf{S}_L^{(1)}$ to the last $\mathbf{S}_L^{(P)}$ and the details are described as follows.

- When $D < |\mathbf{S}_L^{(1)}|$, only $\mathbf{S}_L^{(1)}$ is required to embed D. Then
$$\begin{aligned} D_I^{(1)} &= D; \\ D_I^{(w)} &= 0 \quad \text{for} \quad w = 2, \cdots, P. \end{aligned}$$
- When $|\mathbf{S}_L^{(1)}| \le D < |\mathbf{S}_L^{(1)}| \cdot |\mathbf{S}_L^{(2)}|$, we should use $\mathbf{S}_L^{(1)}$ and $\mathbf{S}_L^{(2)}$ to embed D. Then
$$\begin{aligned} D_I^{(1)} &= \mod(D, |\mathbf{S}_L^{(1)}|); \\ D_I^{(2)} &= \left\lfloor D/|\mathbf{S}_L^{(1)}| \right\rfloor; \\ D_I^{(w)} &= 0 \quad \text{for} \quad w = 3, \cdots, P. \end{aligned}$$
- When $\prod_{v=1}^{u-1} |\mathbf{S}_L^{(v)}| \le D < \prod_{v=1}^{u} |\mathbf{S}_L^{(v)}|$, we should use $\mathbf{S}_L^{(1)}, \cdots, \mathbf{S}_L^{(u)}$ to embed D. Then
$$\begin{aligned} D_I^{(1)} &= \mod(D, |\mathbf{S}_L^{(1)}|); \\ D_I^{(2)} &= \mod\left(\left\lfloor D/|\mathbf{S}_L^{(1)}| \right\rfloor, |\mathbf{S}_L^{(2)}|\right); \\ D_I^{(u-1)} &= \mod\left(\left\lfloor D/(\prod_{v=1}^{u-2} |\mathbf{S}_L^{(v)}|) \right\rfloor, |\mathbf{S}_L^{(u-1)}|\right); \\ D_I^{(u)} &= \left\lfloor D/(\prod_{v=1}^{u-1} |\mathbf{S}_L^{(v)}|) \right\rfloor. \end{aligned}$$

The rest $D_I^{(w)} = 0$ for $w = u+1, \cdots, P$. Note that if $u = P$, all the P blocks are used to embed data.

To better understand the process of calculating $D_I^{(1)}, \cdots, D_I^{(P)}$, we provide a simple example using two multisets $\mathbf{S}_L^{(1)} = \{4, 3, 2, 1\}$ and $\mathbf{S}_L^{(2)} = \{3, 3, 2, 1\}$ with $P = 2$. We first calculate $|\mathbf{S}_L^{(1)}| = 24$ and $|S_L^{(2)}| = 12$ and then obtain the embedding capacity of two blocks $C' = \lfloor \log_2(24 \cdot 12) \rfloor = 8$ according to Eq. (11). Suppose that an 8-bit stream is embedded and then obtain its decimal value $D = 166$. Because $D \in [|\mathbf{S}_L^{(1)}|, |\mathbf{S}_L^{(1)}| \cdot |\mathbf{S}_L^{(2)}|)$, we can obtain $D_I^{(1)} = \mod(D, |\mathbf{S}_L^{(1)}|) = 22$ and $D_I^{(2)} = \left\lfloor D/|\mathbf{S}_L^{(1)}| \right\rfloor = 6$.

In the data extraction process, we can recover D using $D_I^{(1)}, \cdots, D_I^{(P)}$ as
$$D = D_I^{(P)} \prod_{v=1}^{P-1} |\mathbf{S}_L^{(v)}| + \cdots + D_I^{(u)} \prod_{v=1}^{u-1} |\mathbf{S}_L^{(v)}| + \cdots + D_I^{(1)}. \quad (12)$$

Finally, the total embedding capacity can be calculated as
$$C_{total} = \sum_{v=1}^{\lfloor N_e/P \rfloor} C_v{}' + C'_{last}, \quad (13)$$
where N_e is the total number of embeddable blocks, C'_v is the capacity of the v-th combination of P blocks, and C'_{last} is the capacity of the remaining several blocks smaller than P.

3) Entire Data Embedding Process: After determining the embeddable blocks and embedding strategy, we can embed the additional data into the encrypted JPEG bitstream.

First, we use a marker to indicate whether a block is embeddable. In particular, an **embeddable block** is marked "1," whereas a **failed block** is marked as "0." In addition, if the number of AC triples within a block is less than M, there is no need to mark this block and we call a **useless block**. This is because we know that a block cannot be used from the number of AC triples in the block. We embed the marker of each block by replacing the first bit of $ACA^{(1)}$. Then, the first AC coefficients of all marked blocks, **FB**, should be embedded into the image. The length of **FB** is affected by QF and the JPEG image itself. In general, when $QF = 80$, the length of **FB** is approximately 3000 bits. Because the length of **FB** is easy to be calculated after determining the number of useless blocks, it does not need to be embedded into the image.

When embedding the confidential data into the image, one first encrypts the data, calculates the verification information of the embedded data using some Hash functions such as MD5, and finally embeds the verification information along with the embedded data.The final embedding bitstream is
$$\mathbf{ED} = \mathbf{FB}||\mathbf{D}'. \quad (14)$$

Finally, we describe the data-embedding process as follows.

- *Step 1*: Extract the first AC coefficients of all the embeddable and failed blocks to obtain a stream **FB**. Place a marker '1' for an embeddable block, and '0' for a failed block by replacing the first AC coefficient of the block;
- *Step 2*: Encrypt additional data $\mathbf{D}$ by $\mathbf{D}' = \text{Enc}(\mathbf{D}, Key_{ed})$ if it is confidential. Then the final embedding bitstream $\mathbf{ED} = \mathbf{FB}||\mathbf{D}'$;
- *Step 3*: Find P embeddable blocks sequentially, decode the bitstreams and calculate $\mathbf{S}_L^{(1)}, \mathbf{S}_L^{(2)}, \cdots, \mathbf{S}_L^{(P)}$;
- *Step 4*: Calculate their embedding capacity C' using Eq. (11). Take C' bits from the current embedded bitstream **ED** and convert them into a decimal numbers D. Calculate the embedded numbers of the P blocks as $D_I^{(1)}, D_I^{(2)}, \cdots, D_I^{(P)}$.
- *Step 5*: Embed the numbers $D_I^{(1)}, D_I^{(2)}, \cdots, D_I^{(P)}$ using the $\mathbf{S}_L^{(1)}, \mathbf{S}_L^{(2)}, \cdots, \mathbf{S}_L^{(P)}$, respectively, and obtain P permuted multisets of number label $\mathbf{S}_L^{(1)'}, \mathbf{S}_L^{(2)'}, \cdots, \mathbf{S}_L^{(P)'}$, according to the permutation-based embedding method in Section III;
- *Step 6*: Change the order of the AC triples according to the index of $\mathbf{S}_L^{(1)'}, \mathbf{S}_L^{(2)'}, \cdots, \mathbf{S}_L^{(P)'}$. Exchange the marker of the block with the first bit of the new first AC coefficient. Encode the triples as bitstreams;
- *Step 7*: Repeat *Steps 3-6* until all the data have been embedded.

To illustrate the data embedding process, we provide a numeral example in Fig. 5. Suppose that $M = 4$ and that an image block $\mathbf{MCU} = \{DCC, ACC^{(1)}, \cdots, ACC^{(8)}, EOB\}$. After Huffman decoding, we obtain the AC coefficients as $\mathbf{B} = \{p^{(1)}, \cdots, p^{(8)}\} = \{(0, 3, 111_2), \cdots, (4, 1, 1_2)\}$. Then we can calculate the multiset of four label numbers as $\mathbf{S}_L = \{L^{(1)}, L^{(2)}, L^{(3)}, L^{(4)}\} = \{13, 5, 0.58, 0.53\}$ using Eq. (9).

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

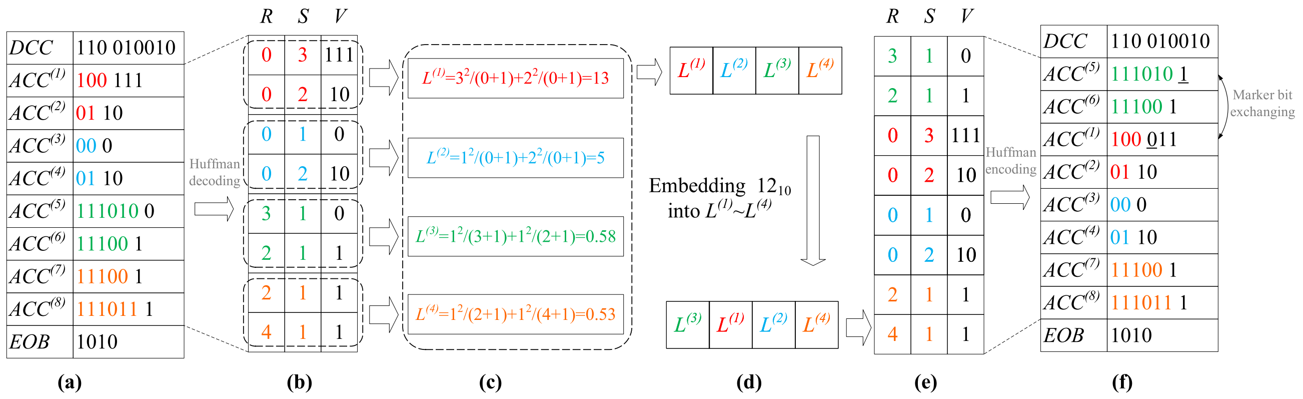

Fig. 5. Numeral example of data embedding process. (a) A block of the JPEG bitstream; (b) dividing the 8 triples of the AC coefficients as 4 groups; (c) label numbers of each group; (d) data embedding; (e) changing the orders of the triple groups according to the index of the permuted label numbers; and (f) Huffman encoding and exchanging the marker of the block with the first bit of the new first AC coefficient.

As described in Section III-A, there are four different numbers in the multiset $\mathbf{S}_L$; thus, $N = 4$ and $K = 4$. As described in Section III-B, we can number each permutation of multiset $\mathbf{S}_L$. When embedding the value 12, we change the multiset $\mathbf{S}_L$ to be the new sequence, which is numbered 12. According to the new sequence, the AC coefficients are rearranged and encoded as a JPEG bitstream. Evidently, the data structures of the triples did not change. Thus, this embedding operation can preserve the file size and format.

C. Data Extraction and Image Recovery

The data extraction and image recovery are separable.

1) Data Extraction: First, we determine the embeddable blocks using the marker and the number of AC triples within each block. Data extraction is performed over P embeddable blocks. After calculating the multisets of label numbers $\mathbf{S}_L^{(1)'}, \mathbf{S}_L^{(2)'}, \cdots, \mathbf{S}_L^{(P)'}$ of the P embeddable blocks, we can obtain the capacity of the P embeddable blocks C' using Eq. (11). Subsequently, the embedded numbers $D_I^{(1)}, D_I^{(2)}, \cdots, D_I^{(P)}$ of the P blocks can be calculated using the **Fast Search** method in Section III-B. Then the embedded number D can be calculated using Eq. (12).

Then D is converted into C'-bit data. After extracting all embeddable blocks, we obtain **ED**. The length of **BF** is the sum of the number of embeddable blocks and failed blocks. After splitting **ED**, we obtain **FB** and **D**$'$. Finally, we decrypt **D**$'$ to obtain the embedded data **D** using the embedding key. The data extraction process is as follows:

- *Step 1*: Calculate the length of **FB** by subtracting the number of useless blocks using the total block number;
- *Step 2*: Find continuous P embeddable blocks according to their markers. Decode the bitstream and calculate their label numbers $\mathbf{S}_L^{(1)'}, \mathbf{S}_L^{(2)'}, \cdots, \mathbf{S}_L^{(P)'}$;
- *Step 3*: Calculate the embedding capacity C' using Eq. (11). Calculate the embedded numbers of the P blocks $D_I^{(1)}, D_I^{(2)}, \cdots, D_I^{(P)}$;
- *Step 4*: Calculate the embedded decimal number D by Eq. (12). Convert D to C'-bit binary numbers;
- *Step 5*: Repeat *Steps 2-4* until all the embedded data **ED** have been extracted. Split **ED** into **FB** and **D**$'$ according to the length of **FB**;
- *Step 6*: Exclude **FB** from **ED** to obtain the embedded data. Decrypt the extracted data using the embedding key if necessary.

2) Image Recovery: The image content was changed by image encryption and data embedding. Thus, we should first recover the changed content during data embedding and then decrypt the image using the encryption key. Because the image recovery must also know the type of each block, **FB** should be first extracted, and the first five steps in the image recovery are exactly the same as with the data extraction. After extracting **FB**, the following operations are performed:

- *Step 1*: Rearrange AC coefficients of the embeddable blocks in descending order and exchange the marker bit. Encode the bitstream and replace the first bits of $ACA^{(1)}$ with **FB**;
- *Step 2*: Decrypt the image using the decryption processes mentioned in Section IV-A, and finally obtain the original JPEG bitstream.

If we first recover the image and then extract the embedded data, we should retain the following information during image recovery: intra block orders of ACHs, inter block orders of ACHs and markers of the blocks. The inter block orders can be obtained using the key that is the hash of $\mathbf{B}_{ACA}$ and thus does not need to preserve. The inter block orders should be preserved, which causes distortion to the recovered image. The markers are retained by replacing the least significant bit (LSB) of DC, which causes dataloss to the recovered image. With the above retained information in the recovered image, one can then completely extract the embedded data. Note that the image distortion can be recovered after data extraction.

V. Experiment Results

In this section, we first analyze how to determine the embeddable blocks and the number of combined blocks, and then simulate our RDH scheme over encrypted JPEG bitstreams, and finally discusses its properties.

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

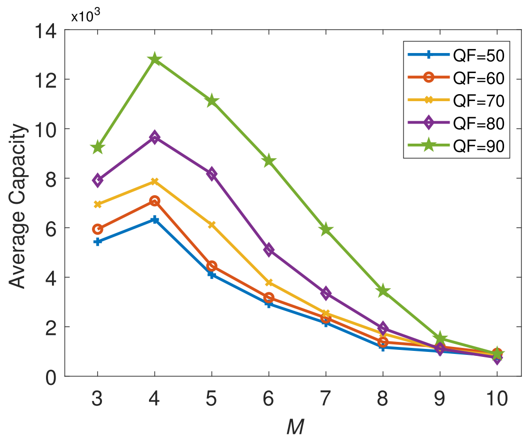

Fig. 6. Average embedding capacity of 100 images under different settings of group number M. The quality factors $QF = 50, 60, 70, 80, 90$.

A. Embeddable Blocks Analysis

Because our permutation-based embedding technique works over an ordered sequence and not all blocks have strict orders for Rs and Ss of the AC coefficients, we use the grouping strategy to enhance the number of embeddable blocks. Evidently, a smaller group number M can lead to a greater number of embeddable blocks. However, with smaller group number, the element number of the ordered sequence is also smaller, causing a lower embedding capacity in each block. Thus, the group number M should be set to an appropriate value.

To obtain the best setting for the group number M, we calculate the average embedding capacity of 100 images in the encrypted domain. These images are randomly selected from the *Bossbase* image database [36] and have the size 512×512. For each image, we encoded it using different quality factors $QF = 50, 60, 70, 80, 90$, and then encrypted these images using the encryption method. The range of M in this experiment was $[3, 10]$. The average embedding capacity of these 100 images under different quality factors are calculated and Fig. 6 shows the results. Evidently, when M is greater than 5, the average embedding capacity decreases rapidly. This is because the number of triples in each group is extremely small when the value of M is relatively large. The largest average embedding capacity can be obtained when $M = 4$ for different quality factors. We further calculated the number of images that can achieve the maximum embedding capacity under different settings of group number M and Fig. 7 show the results for different quality factors. We found that only three settings of M can achieve the maximum embedding capacity: $3, 4, 5$. In addition, most images exhibit the maximum embedding capacity for $M = 4$. Therefore, we set $M = 4$ in the proposed scheme.

B. Number of Combined Blocks

The number of combined blocks P also affects the embedding capacity. Table III lists the embedding capacities of different encrypted images under different settings of P when $M = 4$. Because of the label number calculation method in Eq. (9), almost all embeddable blocks have a strict order,

TABLE III

EMBEDDING CAPACITY (BIT) OF FOUR ENCRYPTED IMAGES WITH DIFFERENT NUMBERS OF COMBINED BLOCKS P ($QF = 80$, $M = 4$)

Images	$P=1$	$P=2$	$P=3$	$P=4$	$P=5$	$P=6$
Baboon	7875	11315	10743	11315	10973	11315
Boat	5092	7710	7295	7710	7473	7710
Couple	3544	5737	5379	5737	5529	5737
Lena	3759	5993	5641	5993	5791	5993

which means that the multiset of label numbers $\mathbf{S}_L^{(i)}$ has four different label numbers. According to Eq. (11), for P combined embeddable blocks, the average embedding capacity of each block is

$$\mu(P) = \frac{\lfloor \sum_{\upsilon=1}^{P} \log_2 |\mathbf{S}_L^{(\upsilon)}| \rfloor}{P}. \tag{15}$$

When $\mathbf{S}_L^{(i)}$ has four different label numbers, its permutation number is $|\mathbf{S}_L^{(i)}| = \frac{4!}{1!1!1!1!} = 24$. Then we can get that $\mu(2) = \mu(4) = \mu(6)$, indicating the same average embedding capacity for $P = 2$, $P = 4$ and $P = 6$. Because a higher computational cost is required for a larger P, we set $P = 2$ in our scheme.

Note that the maximum embedding capacity may be a little different for encrypted and unencrypted JPEG bitstreams, because the encryption will change the block positions and result in different combing results.

C. Simulation Results

Fig. 8 shows the simulation results obtained using images *Lena*, *Pepper*, and *Baboon*. The image owner encrypts the plaintext JPEG images into encrypted JPEG images and sends them to data hider. The data hider can embed additional data (i.e., the label of the image) into the encrypted JPEG images to generate marked encrypted images. The receiver can correctly extract the embedded data and recover the original images.

VI. PERFORMANCE EVALUATIONS

In this section, we present a comprehensive evaluation of the proposed RDH scheme and compare it with other schemes.

A. Embedding Capacity

Because our RDH scheme can preserve the file size and JPEG format, we compared its embedding capacity with the schemes in [29], [30], and [23], which are the latest RDH schemes over encrypted JPEG images that can also preserve the file size and JPEG format. Table IV lists the embedding capacities of several classical images under different quality factors. The BPNZ value indicates the embedded bits per non-zero AC coefficient. In addition, we combine the embedding methods in [23] and [34] with our encryption, which are denoted as "OurEnc+Ref. [23]" and "OurEnc+Ref. [34]", respectively. For most schemes, the embedding capacity increases with the increment of the quality factor. This is because the data are embedded using the non-zero AC coefficients, and the quantized coefficient matrices can remain more non-zero AC coefficients with a higher quality factor.

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

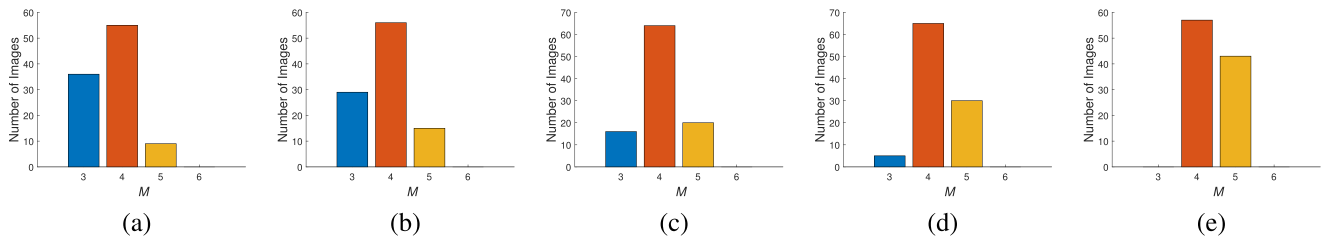

Fig. 7. Image numbers that can achieve the maximum embedding capacity under different settings of group number M when $M \in [3, 10]$: (a) $QF = 50$, (b) $QF = 60$, (c) $QF = 70$, (d) $QF = 80$, and (e) $QF = 90$.

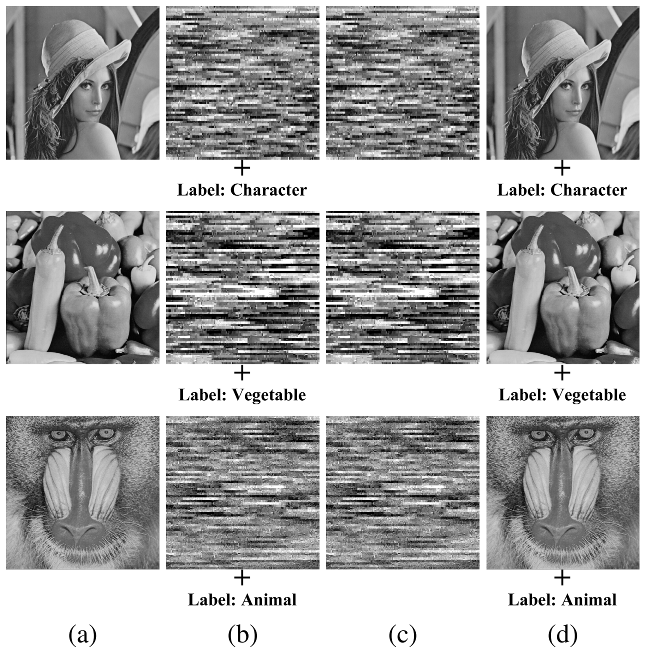

Fig. 8. (a) The JPEG images *Lena*, *Pepper* and *Baboon*; (b) The encrypted images and their labels of images; (c) The marked encrypted images after embedding their labels; (d) The recovered images and extracted labels of images.

As shown in Table IV, our scheme can embed many more bits and the BPNZ is much larger than that of other schemes under the same quality factors. Its average embedding bits are approximately twice those of the previous best scheme in [23]. The embedding capacity in our scheme is the effective embedding capacity, which indicates the number of bits that a data hider can embed. Table V shows the embedding capacity of four images with the full payload and the PSNRs between the recovered images and original images. All the PSNRs are $+\infty$, which indicate the reversibility of our scheme.

B. File Size Preservation

Our RDH scheme can obtain encrypted JPEG bitstreams with both file size and format preservation, because the encryption and data embedding processes don not destroy the data structure of the JPEG format. Table VI lists the file size change of several RDH schemes for embedded data of several lengths. The scheme in [24] can generate marked encrypted JPEG bitstream with a reduced size because it compresses some DC and AC coefficients. However, this operation destroys the JPEG format, changes the file information, and augments the computational complexity of image recovery. In addition, it can only achieve a small embedding capacity [24]. The marked encrypted JPEG bitstream obtained by the scheme in [25] is larger than the original JPEG bitstream, and its size increases with the payload increment. This is because some data are embedded in the JPEG header, destroying the JPEG format. Among all previous schemes with both file size and format preservation, the scheme in [23] achieves the largest embedding capacity. However, our RDH scheme achieves a much larger embedding capacity than the scheme in [23].

Remark: The marked encrypted JPEG bitstreams in our RDH scheme and the scheme in [23] have slightly different file sizes from the original JPEG bitstream, which are caused by byte alignment rather than encryption and data embedding. According to the discussions in [19] and [32], this kind of file size change does not increase with the increase in payload and is ignored in all RDH schemes over encrypted JPEG bitstreams [23], [29], [30].

To further verify the large embedding capacity and file size preservation of our scheme, we randomly selected 200 images from the *Bossbase* dataset and show their average embedding capacity and file size change with the full payload of our scheme in Table VII. Our scheme achieved a large embedding capacity and file size preservation for these images.

C. Format Compatibility

Many RDH schemes over encrypted JPEG bitstreams cannot preserve the JPEG format in marked encrypted JPEG bitstreams. The schemes in [24] and [25] use an exclusive-or-based encryption strategy to encrypt the DC coefficients, which may cause an overflow to the quantized DC coefficients, according to the discussions in [32]. These schemes also change the file information, thereby making the size of the encrypted image different from that of the original image. For the scheme in [31], the characteristics of Huffman coding are used for data embedding, which destroys the JPEG format and makes the encrypted JPEG bitstreams unreadable to JPEG decoders. In our proposed scheme, the encryption and data embedding processes do not destroy the data structure of the JPEG format. Thus, our scheme can preserve the JPEG format and the encrypted JPEG bitstreams and the marked encrypted JPEG bitstreams are completely compatible with all JPEG decoders.

D. Security Analysis

In our RDH scheme, we use the encryption method introduced in [32] to encrypt the DC coefficients. The AC

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

TABLE IV

EMBEDDING CAPACITY OF DIFFERENT RDH SCHEMES WITH FILE SIZE AND FORMAT PRESERVATION UNDER DIFFERENT QUALITY FACTORS. THE BOLD FONT INDICATES THE BEST PERFORMANCE

Quality Factor		$QF = 50$		$QF = 60$		$QF = 70$		$QF = 80$		$QF = 90$	
Embedding Capacity		Bits	BPNZ	Bits	BPNZ	Bits	BPNZ	Bits	BPNZ	Bits	BPNZ
Baboon	Ref. [29]	775	0.012	775	0.010	775	0.009	775	0.007	775	0.005
	Ref. [30]	1238	0.019	1277	0.017	1435	0.016	1723	0.016	2044	0.013
	Ref. [23]	3775	0.058	4472	0.060	5311	0.060	6344	0.058	6520	0.043
	OurEnc+Ref. [23]	3303	0.050	3972	0.053	4957	0.056	5943	0.054	6292	0.041
	OurEnc+Ref. [34]	4387	0.067	5239	0.067	6148	0.069	7153	0.065	8589	0.056
	Ours	**6581**	**0.100**	**7433**	**0.099**	**8888**	**0.100**	**11315**	**0.103**	**13477**	**0.088**
Boat	Ref. [29]	726	0.021	750	0.019	768	0.016	775	0.013	775	0.008
	Ref. [30]	733	0.021	797	0.020	834	0.017	906	0.015	1475	0.016
	Ref. [23]	1728	0.050	2173	0.054	2726	0.056	4332	0.070	6913	0.073
	OurEnc+Ref. [23]	1267	0.037	1755	0.044	2023	0.042	3040	0.049	5580	0.058
	OurEnc+Ref. [34]	2540	0.073	3043	0.076	3832	0.079	5368	0.086	7660	0.080
	Ours	**3022**	**0.086**	**3736**	**0.092**	**4963**	**0.102**	**7710**	**0.123**	**11782**	**0.123**
Couple	Ref. [29]	757	0.023	765	0.020	772	0.017	775	0.013	775	0.009
	Ref. [30]	582	0.018	718	0.019	929	0.020	1037	0.018	1341	0.016
	Ref. [23]	1184	0.035	1601	0.042	2347	0.051	3107	0.053	4772	0.056
	OurEnc+Ref. [23]	1051	0.032	1391	0.036	2173	0.047	2658	0.046	4490	0.052
	OurEnc+Ref. [34]	1653	0.050	2032	0.053	2819	0.060	4237	0.073	6568	0.077
	Ours	**1931**	**0.057**	**2740**	**0.071**	**3950**	**0.085**	**5737**	**0.099**	**9327**	**0.109**
Lena	Ref. [29]	735	0.029	756	0.025	771	0.021	775	0.016	775	0.010
	Ref. [30]	523	0.020	570	0.019	684	0.018	894	0.018	1589	0.020
	Ref. [23]	889	0.035	1171	0.039	1843	0.049	2762	0.056	5660	0.073
	OurEnc+Ref. [23]	820	0.032	971	0.032	1541	0.041	2425	0.049	4826	0.062
	OurEnc+Ref. [34]	1577	0.062	1975	0.067	2679	0.072	4134	0.084	7082	0.090
	Ours	**1973**	**0.077**	**2588**	**0.086**	**3966**	**0.106**	**5993**	**0.122**	**9997**	**0.127**

TABLE V

EMBEDDING CAPACITY (BIT) OF OUR RDH SCHEME WITH THE FULL PAYLOAD UNDER DIFFERENT QUALITY FACTORS AND THE PSNRS (dB) BETWEEN THE RECOVERED IMAGES AND ORIGINAL IMAGES

	Quality Factor					
	50		70		90	
Image	Bits	PSNR	Bits	PSNR	Bits	PSNR
Baboon	6581	$+\infty$	8888	$+\infty$	13477	$+\infty$
Boat	3022	$+\infty$	4963	$+\infty$	11782	$+\infty$
Couple	1931	$+\infty$	3950	$+\infty$	9327	$+\infty$
Lena	1973	$+\infty$	3966	$+\infty$	9997	$+\infty$

coefficients are divided into ACAs and ACHs. The ACAs are encrypted using AES with an encryption key, whereas the ACHs are encrypted by a scrambling method.

Our encryption system is a plaintext-related encryption system, because the encryption keys of the DC coefficients and ACHs are generated using a hash function for the ACAs, and the ACAs are different in different images. Any change to the plaintext results in different encryption keys for encrypting the DC coefficients and ACHs. Thus, the encryption system can resist chosen-plaintext attacks.

1) Security of DC Coefficients: The method introduced in [32] was used to encrypt the DC coefficients. According to the discussions in [32], the space is

$$\Phi_1 = \prod_{t=1}^{T} Y_t! \times \prod_{l=1}^{Z} 2^{\lfloor N/(2l) \times W_l \rfloor}, \tag{16}$$

where $\mathbf{G_t}$ $(t = 1, 2, \cdots, T)$ is the t-th group of consecutive DCCs with the same sign, Y_t denotes the number of DCCs in $\mathbf{G_t}$, Z is the number of iterations, N is the number of all DCCs, and W_j denotes the percentage of the groups that can be exchanged.

2) Security of AC Coefficients: The two parts of the AC coefficients are encrypted using different strategies. The ACAs are encrypted using AES with an encryption key and AES has high security for protecting confidentiality. Thus, the ACAs can be effectively protected.

The ACHs are encrypted using a scrambling method and the key is generated using a hash function to the ACAs. For an image with size $H \times W$, the space for scrambling is

$$\Phi_2 = (H \times W/64)!. \tag{17}$$

Some image information may be hidden in the file structure, such as the number of non zero AC coefficients, and this information may partially reconstruct the edges and textures of the original image [37]. According to the discussions in [38], a JPEG encryption system should be highly secure to resist this outline attack. In our experiments, we used the energy of

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

TABLE VI
FILE SIZE CHANGE COMPARISONS UNDER SEVERAL LENGTHS OF EMBEDDED DATA D

Payload (bits)		$D = 250$		$D = 500$		$D = 1000$		$D = 1500$		$D = 2500$	
File Size Change		Bits	Ratio(%)	Bits	Ratio(%)	Bits	Ratio(%)	Bits	Ratio(%)	Bits	Ratio(%)
Baboon	Ref. [24]	-1224	-0.19	-1224	-0.19	-	-	-	-	-	-
	Ref. [25]	880	0.14	952	0.15	1120	0.17	1360	0.21	1808	0.28
	Ref. [23]	360	0.06	360	0.06	352	0.05	384	0.06	392	0.06
	Ours	336	0.05	336	0.05	336	0.05	368	0.06	424	0.08
Boat	Ref. [24]	-2128	-0.54	-2128	-0.54	-	-	-	-	-	-
	Ref. [25]	792	0.20	920	0.23	1104	0.28	1320	0.34	1504	0.38
	Ref. [23]	552	0.14	512	0.13	512	0.13	480	0.12	576	0.15
	Ours	504	0.13	504	0.13	504	0.13	504	0.13	566	0.15
Couple	Ref. [24]	-2056	-0.56	-2056	-0.56	-	-	-	-	-	-
	Ref. [25]	1072	0.29	1208	0.33	1368	0.37	1536	0.42	1976	0.54
	Ref. [23]	208	0.06	256	0.07	248	0.07	248	0.07	240	0.07
	Ours	240	0.07	248	0.07	248	0.08	264	0.07	296	0.08
Lena	Ref. [24]	-696	-0.22	-696	-0.22	-	-	-	-	-	-
	Ref. [25]	1176	0.37	1264	0.40	1608	0.51	1920	0.61	2176	0.69
	Ref. [23]	240	0.08	216	0.07	232	0.07	152	0.05	80	0.03
	Ours	160	0.05	160	0.05	168	0.05	184	0.06	192	0.06

TABLE VII
AVERAGE EMBEDDING CAPACITY AND FILE SIZE CHANGE WITH THE FULL PAYLOAD OF OUR SCHEME ON 200 RANDOMLY SELECTED IMAGES FROM THE *Bossbase* DATASET

Quality Factor	Capacity		File Size Change	
	Bits	BPNZ	Bits	Ratio(%)
50	3041.2	0.079	184.3	0.06
60	3932.9	0.089	234.7	0.06
70	4667.2	0.096	251.2	0.07
80	7536.1	0.103	292.4	0.07
90	10937.2	0.111	332.5	0.07

AC coefficients introduced in [38] to evaluate the security of our scheme resisting an outline attack. We can calculate the energy of the AC coefficients in i-th block as

$$f_i = \left(\sum_{x=1}^{8}\sum_{y=1}^{8}|DCT_i(x, y)|\right) - |DCT_i(1, 1)|. \tag{18}$$

The total number of N results $\{f_1, f_2, f_3, \cdots, f_N\}$ is obtained from N blocks. Normalization to obtain $\{f_1', f_2', f_3', \cdots, f_N'\}$. Then an attack image of the same size as the original image can be built and all pixels of the i-th block in the attack image are set as $\lfloor f_i' \times 255 \rfloor$. Fig. 9 shows the outline attack of the proposed RDH scheme using image *Lena*. As can be seen, the built attack image cannot reveal any information regarding the original image. According to the discussions in [38], these results can demonstrate the high security of the AC coefficients under this attack.

3) Security of Embedded Data: The security of embedded data is also important [39]. It is impossible for an attacker to obtain the embedded data when the embedded data are encrypted. Besides, we embed verification information along with the embedded data, which can prevent attacker modifying

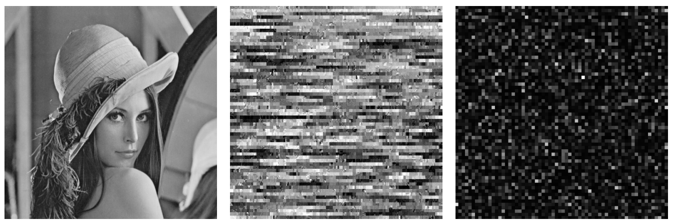
(a) (b) (c)

Fig. 9. Outline attack results: (a) The image *Lena*, (b) the encryption result, and (c) the built attacked image.

or removing the embedded data. After receiving the image, the receiver verifies whether the embedded data is correct and complete using the verification information. The embedded encrypted image needs to be resent if the verification fails.

4) Discussions: Because a JPEG bitstream has a special structure, treating it as a binary stream and encrypting it using an existing encryption standard directly destroys its format, making it unreadable to JPEG decoders. Therefore, special encryption strategies should be developed for JPEG bitstreams.

Many encryption schemes only perform scrambling operations on the DC and AC coefficients, which cannot result in a high encryption performance [23]. In our scheme, the DC coefficients are encrypted using the encryption method introduced in [32] with a high performance and the ACAs are encrypted using AES. Because the security keys of DC and ACH are dependent on the ACAs, the encrypted strategies are sensitive to plaintext. These operations can guarantee a better encryption performance than many previous methods. However, a strict security analysis of the JPEG encryption methods should be further explored.

E. Advantages of Our Scheme

In our RDH over encrypted JPEG bitstreams, the original JPEG bitstream is encrypted using the previous JPEG

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

encryption and AES, while additional data are embedded using a large-capacity permutation-based embedding technique. Besides, a grouping method is used to enhance the number of embeddable blocks. With these measures, our RDH scheme can achieve the following advantages:

(1) Our RDH scheme can ensure image content confidentiality, because the used previous encryption strategy for DC coefficients is proved to have a high security level [32], and the AES is used to encrypt the ACAs.
(2) It can achieve a large embedding capacity, since the data embedding technique has a high efficiency to embed data and the grouping method can greatly enhance the number of embeddable blocks.
(3) Our RDH scheme can obtain the encrypted JPEG bitstreams with both file size and format preservation, because the encryption and data embedding processes don't destroy the data structure of JPEG format.
(4) The data extraction and image recovery are completely separable, and the operations are commutative. This is suitable for many applications.
(5) There is no pre-processing for image owners. The image owners can encrypt their images using different encryption techniques. However, the encryption techniques cannot change the order of the ACHs within each block. Otherwise, the permutation-based embedding will be ineffective.

VII. Conclusion

In this study, we propose an RDH scheme over encrypted JPEG bitstreams to securely store and transmit JPEG bitstreams. We first propose a permutation-based embedding technique on an ordered sequence, which can embed more data than previous data-embedding techniques. Using the proposed embedding technique, we designed an RDH scheme over JPEG bitstreams with a large embedding capacity. A grouping method was used to boost the number of embeddable blocks in the JPEG bitstreams. Because the encryption and data embedding processes do not destroy the data structure of the JPEG format, the proposed RDH scheme can preserve both the file size and JPEG format. Performance evaluations show that it ensures the image confidentiality and achieves a much larger embedding capacity than existing schemes.

References

[1] P. Puteaux and W. Puech, "An efficient MSB prediction-based method for high-capacity reversible data hiding in encrypted images," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 7, pp. 1670–1681, Jul. 2018.
[2] K. Ma, W. Zhang, X. Zhao, N. Yu, and F. Li, "Reversible data hiding in encrypted images by reserving room before encryption," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 3, pp. 553–562, Mar. 2013.
[3] W. Zhang, K. Ma, and N. Yu, "Reversibility improved data hiding in encrypted images," *Signal Process.*, vol. 94, pp. 118–127, Jan. 2014.
[4] X. Cao, L. Du, X. Wei, D. Meng, and X. Guo, "High capacity reversible data hiding in encrypted images by patch-level sparse representation," *IEEE Trans. Cybern.*, vol. 46, no. 5, pp. 1132–1143, May 2016.
[5] X. Zhang, "Reversible data hiding in encrypted image," *IEEE Signal Process. Lett.*, vol. 18, no. 4, pp. 255–258, Apr. 2011.
[6] X. Zhang, "Separable reversible data hiding in encrypted image," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 2, pp. 826–832, Apr. 2012.
[7] J. Zhou, W. Sun, L. Dong, X. Liu, O. C. Au, and Y. Y. Tang, "Secure reversible image data hiding over encrypted domain via key modulation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 3, pp. 441–452, Mar. 2016.
[8] Y.-Q. Shi, X. Li, X. Zhang, H.-T. Wu, and B. Ma, "Reversible data hiding: Advances in the past two decades," *IEEE Access*, vol. 4, pp. 3210–3237, 2016.
[9] X. Liao, K. Li, and J. Yin, "Separable data hiding in encrypted image based on compressive sensing and discrete Fourier transform," *Multimedia Tools Appl.*, vol. 76, no. 20, pp. 20739–20753, Oct. 2017.
[10] Z. Liu and C.-M. Pun, "Reversible data hiding in encrypted images using chunk encryption and redundancy matrix representation," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 2, pp. 1382–1394, Mar. 2022, doi: 10.1109/TDSC.2020.3011838.
[11] Y. Wang and W. He, "High capacity reversible data hiding in encrypted image based on adaptive MSB prediction," *IEEE Trans. Multimedia*, vol. 24, pp. 1288–1298, 2022, doi: 10.1109/TMM.2021.3062699.
[12] R. Bhardwaj and A. Aggarwal, "An improved block based joint reversible data hiding in encrypted images by symmetric cryptosystem," *Pattern Recognit. Lett.*, vol. 139, pp. 60–68, Nov. 2020.
[13] Q. Lingfeng, C. Fan, Z. Shanjun, and H. He, "Cryptanalysis of reversible data hiding in encrypted images by block permutation and co-modulation," *IEEE Trans. Multimedia*, vol. 24, pp. 2924–2937, 2022, doi: 10.1109/TMM.2021.3090588.
[14] F. Khelifi, "On the security of a stream cipher in reversible data hiding schemes operating in the encrypted domain," *Signal Process.*, vol. 143, pp. 336–345, Feb. 2018.
[15] Y. Ke, M.-Q. Zhang, J. Liu, T.-T. Su, and X.-Y. Yang, "Fully homomorphic encryption encapsulated difference expansion for reversible data hiding in encrypted domain," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 8, pp. 2353–2365, Aug. 2020.
[16] B. Chen, X. Wu, W. Lu, and H. Ren, "Reversible data hiding in encrypted images with additive and multiplicative public-key homomorphism," *Signal Process.*, vol. 164, pp. 48–57, Nov. 2019.
[17] M. Li and Y. Li, "Histogram shifting in encrypted images with public key cryptosystem for reversible data hiding," *Signal Process.*, vol. 130, pp. 190–196, Jan. 2017.
[18] P. Zheng and J. Huang, "Discrete wavelet transform and data expansion reduction in homomorphic encrypted domain," *IEEE Trans. Image Process.*, vol. 22, no. 6, pp. 2455–2468, Jun. 2013.
[19] G. K. Wallace, "The JPEG still picture compression standard," *Commun. ACM*, vol. 34, no. 4, pp. 30–44, Apr. 1991.
[20] Z. Tang, M. Pang, C. Yu, G. Fan, and X. Zhang, "Reversible data hiding for encrypted image based on adaptive prediction error coding," *IET Image Process.*, vol. 15, no. 11, pp. 2643–2655, Sep. 2021.
[21] C. Yu, X. Zhang, G. Li, S. Zhan, and Z. Tang, "Reversible data hiding with adaptive difference recovery for encrypted images," *Inf. Sci.*, vol. 584, pp. 89–110, Jan. 2022.
[22] C. Yu, X. Zhang, X. Zhang, G. Li, and Z. Tang, "Reversible data hiding with hierarchical embedding for encrypted images," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 2, pp. 451–466, Feb. 2022.
[23] J. He, J. Chen, W. Luo, S. Tang, and J. Huang, "A novel high-capacity reversible data hiding scheme for encrypted JPEG bitstreams," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 12, pp. 3501–3515, Apr. 2019.
[24] Z. Qian, H. Zhou, X. Zhang, and W. Zhang, "Separable reversible data hiding in encrypted JPEG bitstreams," *IEEE Trans. Depend. Secur. Comput.*, vol. 15, no. 6, pp. 1055–1067, Dec. 2018.
[25] Z. Qian, H. Xu, X. Luo, and X. Zhang, "New framework of reversible data hiding in encrypted JPEG bitstreams," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 2, pp. 351–362, Feb. 2019.
[26] Z. Qian and X. Zhang, "Lossless data hiding in JPEG bitstream," *J. Syst. Softw.*, vol. 85, no 2, pp. 309–313, 2012.
[27] Y. Qiu, Z. Qian, H. He, H. Tian, and X. Zhang, "Optimized lossless data hiding in JPEG bitstream and relay transfer-based extension," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 4, pp. 1380–1394, Apr. 2021.
[28] C. Zhang, B. Ou, H. Tian, and Z. Qin, "Reversible data hiding in JPEG bitstream using optimal VLC mapping," *J. Vis. Commun. Image Represent.*, vol. 71, Aug. 2020, Art. no. 102821.
[29] Z. Qian, X. Zhang, and S. Wang, "Reversible data hiding in encrypted JPEG bitstream," *IEEE Trans. Multimedia*, vol. 16, no. 5, pp. 1486–1491, Aug. 2014.
[30] J.-C. Chang, Y.-Z. Lu, and H.-L. Wu, "A separable reversible data hiding scheme for encrypted JPEG bitstreams," *Signal Process.*, vol. 133, pp. 135–143, Apr. 2017.

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.

[31] S. Sheidani, A. Mahmoudi-Aznaveh, and Z. Eslami, "CPA-secure privacy-preserving reversible data hiding for JPEG images," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 3647–3661, 2021.

[32] J. He, S. Huang, S. Tang, and J. Huang, "JPEG image encryption with improved format compatibility and file size preservation," *IEEE Trans. Multimedia*, vol. 20, no. 10, pp. 2645–2658, Oct. 2018.

[33] C. Zhang, J. Li, S. Wang, and Z. Wang, "An encrypted medical image retrieval algorithm based on DWT-DCT frequency domain," in *Proc. IEEE 15th Int. Conf. Softw. Eng. Res., Manage. Appl. (SERA)*, Jun. 2017, pp. 135–141.

[34] S. Ong, K. Wong, and K. Tanaka, "Scrambling–embedding for JPEG compressed image," *Signal Process.*, vol. 109, pp. 38–53, Apr. 2015.

[35] J. Long, Z. Yin, J. Lv, and X. Zhang, "Rotation based reversible data hiding for JPEG images," *IETE Tech. Rev.*, vol. 33, no. 6, pp. 607–614, 2016.

[36] P. Bas, T. Filler, and T. Pevný, "'Break our steganographic system': The ins and outs of organizing boss," in *Proc. Int. Workshop Inf. Hiding*. Berlin, Germany: Springer, 2011, pp. 59–70.

[37] W. Li and Y. Yuan, "A leak and its remedy in JPEG image encryption," *Int. J. Comput. Math.*, vol. 84, no. 9, pp. 1367–1378, Sep. 2007.

[38] K. Minemura, Z. Moayed, K. Wong, X. Qi, and K. Tanaka, "JPEG image scrambling without expansion in bitstream size," in *Proc. 19th IEEE Int. Conf. Image Process.*, Sep. 2012, pp. 261–264.

[39] I. C. Dragoi and D. Coltuc, "On the security of reversible data hiding in encrypted images by MSB prediction," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 187–189, 2021.

Zhongyun Hua (Member, IEEE) received the B.S. degree in software engineering from Chongqing University, Chongqing, China, in 2011, and the M.S. and Ph.D. degrees in software engineering from the University of Macau, Macao, China, in 2013 and 2016, respectively.

He is currently an Associate Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. He has published more than 60 papers on the subject, receiving more than 3800 citations. His research interests include chaotic systems, chaos-based applications, data hiding, and multimedia security.

Ziyi Wang received the B.S. degree in computer science and technology from Nankai University, Tianjin, China, in 2016. He is currently pursuing the M.S. degree in computer technology with the Harbin Institute of Technology, Shenzhen, China. His research interests include reversible data hiding in encrypted JPEG bitstreams.

Yifeng Zheng received the Ph.D. degree in computer science from the City University of Hong Kong, Hong Kong, in 2019. He is an Assistant Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. He worked as a Post-Doc with the Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia, and the City University of Hong Kong. His current research interests are focused on security and privacy related to cloud computing, the IoT, machine learning, and multimedia. His work has appeared in prestigious venues such as ESORICS, DSN, ACM AsiaCCS, IEEE INFOCOM, IEEE ICDCS, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, and IEEE TRANSACTIONS ON SERVICES COMPUTING. He received the Best Paper Award in the European Symposium on Research in Computer Security (ESORICS) in 2021.

Yongyong Chen (Member, IEEE) received the B.S. and M.S. degrees from the Shandong University of Science and Technology, Qingdao, China, in 2014 and 2017, respectively, and the Ph.D. degree from the University of Macau, Macao, in 2020. He is currently an Assistant Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. He has published more than 30 research papers in top-tier journals and conferences, including IEEE TRANSACTIONS ON IMAGE PROCESSING, IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, IEEE TRANSACTIONS ON MULTIMEDIA, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, IEEE TRANSACTIONS ON COMPUTATIONAL IMAGING, IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING, *Pattern Recognition*, and ACM MM. His research interests include image processing, data mining, and computer vision.

Yuanman Li (Member, IEEE) received the B.Eng. degree in software engineering from Chongqing University, Chongqing, China, in 2012, and the Ph.D. degree in computer science from the University of Macau, Macao, in 2018.

From 2018 to 2019, he was a Post-Doctoral Fellow with the State Key Laboratory of Internet of Things for Smart City, University of Macau. He is currently an Assistant Professor with the College of Electronics and Information Engineering, Shenzhen University, Shenzhen, China. His current research interests include data representation, multimedia security and forensics, computer vision, and machine learning.

Authorized licensed use limited to: University Town Library of Shenzhen. Downloaded on October 10,2023 at 14:14:52 UTC from IEEE Xplore. Restrictions apply.