Separable Reversible Data Hiding in Encrypted JPEG Bitstreams

Zhenxing Qian, *Member, IEEE*, Hang Zhou, Xinpeng Zhang, *Member, IEEE*, and Weiming Zhang

Abstract—While most techniques of reversible data hiding in encrypted images (RDH-EI) are developed for uncompressed images, this paper provides a separable reversible data hiding protocol for encrypted JPEG bitstreams. We first propose a JPEG encryption algorithm, which enciphers an image to a smaller size and keeps the format compliant to JPEG decoder. After a content owner uploads the encrypted JPEG bitstream to a remote server, a data hider embeds an additional message into the encrypted copy without changing the bitstream size. On the recipient side, the original bitstream can be reconstructed losslessly using an iterative recovery algorithm based on the blocking artifact. Since message extraction and image recovery are separable, anyone who has the embedding key can extract the message from the marked encrypted copy. Experimental results show that the proposed method outperforms a previous work in terms of separation capability, embedding capacity and security.

Index Terms—Reversible data hiding, information hiding, image recovery, JPEG encryption

1 Introduction

Signal processing in encrypted domain (**SPED**) for privacy preserving has attracted considerable research interests in recent years [1]. In cloud computing and delegated calculation, users who are unwilling to reveal contents of the original signal may send an encrypted copy to a remote server. The server has to accomplish signal processing in the encrypted domain [2]. Many approaches have been proposed for different applications, for example, compressing encrypted images [3], signal transformation in ciphertexts [4], pattern recognition in encrypted domain [5], watermarking in encrypted multimedia [6], data searching in encrypted dataset [7], etc. Reversible data hiding in encrypted images (RDH-EI) is another topic of SPED [8].

RHD-EI allows a server to embed additional message into an encrypted image uploaded by the content owner, and guarantees that the original content can be losslessly recovered after decryption on the recipient side. Generally, reversibility is closely related to the embedding payload. If the original image can be losslessly recovered when the payload does not exceed the achievable capacity, we say it is *reversible*. Meanwhile, RDH-EI protocols are always designed for natural images. Since a natural image always contains large smooth areas, i.e., redundancies, one can embed data into the original image and losslessly recover it [8], [9], [10], [11], [12], [13], [14], [15], [14], [17], [18], [19], [20], [21], [22]. Unlike robust watermarking, reversible data hiding are widely used when perfect image reconstruction and data extraction are emphasized while robustness against malicious attacks is not considered [23].

RDH-EI is useful in many applications [8], [9], [10], [11], [12], [13], [14], [15], [14], [17], [18], [19], [20], [21], [22]. For example, in cloud storage as shown in Fig. 1, an image owner may store images in the cloud. Before uploading the images, the owner encrypts the contents to preserve privacy. For management purposes, the cloud administrator can embed labels, such as user information, timestamps and remarks, into the ciphertexts. Therefore, labels are attached inside these ciphertexts, and storage overheads can be saved. The embedded information can also be extracted exactly by the administrator or authorized users. Meanwhile, when an authorized user downloads the encrypted image containing additional message from the cloud, RDH-EI protocol also guarantees that the original content can be losslessly recovered after decryption.

Emerging works on RDH-EI are reviewed in Section 2. While most of the related works are applicable to uncompressed images, this paper focuses on RDH in encrypted JPEG bitstream, the most popular image format, aiming at providing an RDH-EI approach with separable extraction capability, high embedding capacity, and secure encryption. We first propose an encryption scheme for enciphering JPEG bitstreams. Based on JPEG encryption, a reversible data hiding method is developed for service providers to embed additional bits. Finally, we propose an iterative algorithm to recover the original image. In this work, lossless recovery is required. Although JPEG encoding itself is lossy, users always hope not to introduce further degradation to a JPEG image while uploading. That is why lossless recovery is required.

Compared with our previous work of RDH-EI for JPEG bitstreams [12], the present method has three contributions. First, data extraction and image recovery can be separated, while both features in [12] must be realized jointly. Second, a rearrangement and enciphering algorithm is proposed to avoid leaking of image contents, making the present method securer

- *Z. Qian and X. Zhang are with School of Communication and Information Engineering, Shanghai University, Shanghai 200444, China. E-mail: {zxqian, xzhang}@shu.edu.cn.*
- *H. Zhou and W. Zhang are with the School of Information Science and Technology, University of Science and Technology of China, Hefei 230026, China. E-mail: hangzhou_ryan@163.com, zhangwm@ustc.edu.cn.*

Manuscript received 30 May 2016; revised 28 Nov. 2016; accepted 28 Nov. 2016. Date of publication 1 Dec. 2016; date of current version 9 Nov. 2018. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TDSC.2016.2634161

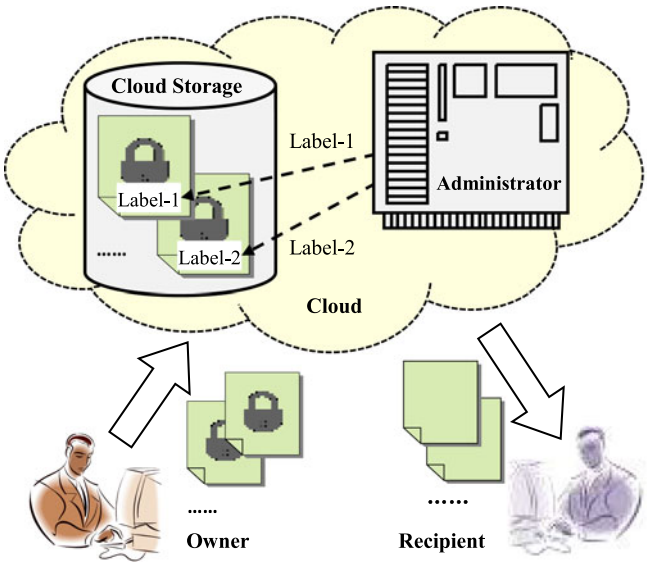

Fig. 1. An example of RDH-EI application.

than the previous JPEG encryption algorithm proposed in [12]. Third, an algorithm for compression and iterative recovery is proposed to reversibly hide data into an encrypted bitstream. As a result, a larger embedding payload is achieved. The rest of the paper is organized as follows. Previous works related to RDH-EI are surveyed in Section 2. The proposed system is developed in Section 3. Section 4 provides experimental results and analyses. The paper is concluded in Section 8.

2 Related Works

Generally, an RDH-EI framework has three parties, content owner, data hider and recipient, as shown in Fig. 2. To preserve privacy, the content owner encrypts an original image using an encryption key, and uploads the encrypted copy to a remote server. On the server side, the data hider embeds additional messages into the encrypted image using an embedding key to generate a marked version. The recipient can losslessly recover the original image using the encryption key after downloading the marked version. There are two different cases: both the data hider and recipient can extract the hidden message; and only the recipient can extract the message, hence two types of technique: *separable* RDH-EI and *joint* RDH-EI.

2.1 Separable RDH-EI

The word *separable* means separating data extraction from image recovery, i.e., additional messages can be extracted directly from the marked encrypted image without revealing the image content. Only those who have the embedding key can extract the messages from a marked encrypted image.

A separable RDH-EI method was first proposed in [13]. The data hider permutes and divides the encrypted pixels into segments, and compresses several LSB-planes of each segment to fewer bits using a pseudo-randomly generated matrix. As a result, spare room in each segment is created to accommodate additional messages. On the recipient side, LSBs of each segment are estimated using the MSBs of the neighboring pixels. After comparing the estimated bits with the extracted vectors, the recipient can recover the original contents. Since the additional message can be extracted directly from LSBs of the encrypted images, data extraction and image recovery are therefore separable. This method was improved in [14] by selecting appropriate bitplanes in the encrypted image, leading to a higher embedding capacity. In [15], distributed source coding (DSC) is used to achieve separable RDH-EI. The data hider compresses some selected bits in the encrypted image to create room for the additional hidden message. In this method, the Slepian-Wolf encoder based on low density parity check (LDPC) is used. With the DSC based embedding, a much higher capacity is obtained.

With a different idea, [16] creates room for embedded data in a plaintext image by embedding LSBs of some pixels into other pixels using traditional RDH for plaintext images. The pre-processed image is then encrypted by the content owner to construct an encrypted image. Positions of these evacuated LSBs in the encrypted image are used to accommodate additional messages. A large payload, up to 0.5 bit-per-pixel, can be achieved. Similarly, another method based on estimation was proposed in [17], in which a large portion of pixels are used to estimate the rest before encryption. Final version of the encrypted image is formulated by concatenating the encrypted estimating errors with the encrypted pixels. On the server side, additional bits are embedded into the encrypted image by modifying the estimation errors. In [18], an RDH-EI method based on patch-level sparse representation was proposed to explore correlations between neighboring pixels. After self-embedding encoded residual errors and a learned dictionary into the original image, the data hider can embed more secret messages into the encrypted image. Another RDH-EI approach was realized using histogram shift and spatial permutation [19]. The method simultaneously prepares room before image encryption and hides data into the encrypted image using histogram modification based RDH. The separable methods proposed in [16], [17], [18], [19], [20], [21] have high embedding rates and good recovery capability. However, they all require extra RDH operations before image encryption, thus contradict the very

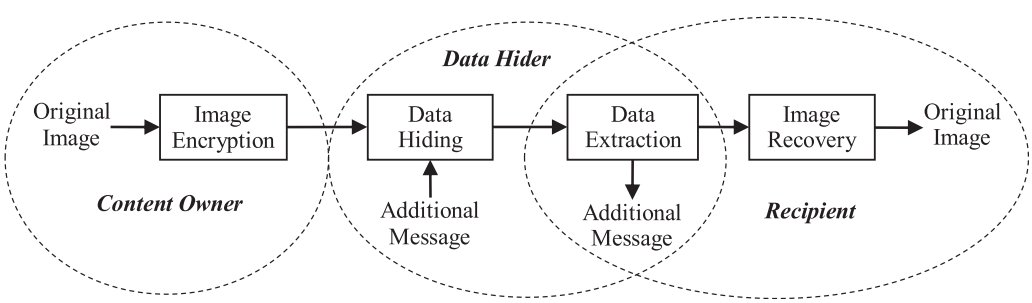

Fig. 2. General framework of RDH-EI.

purpose of RDH-EI, since the extra operations are performed to the plaintext rather than encrypted images.

There are also some interesting works based on commutative encryption and data hiding [24], [25], [26], where *commutative* means that the orders of encryption and data hiding/extraction can be swapped. Although some have separable features, commutative RDH is different from the framework proposed in the present paper.

2.2 Joint RDH-EI

In joint RDH-EI, the additional message can only be extracted by the recipient after image decryption, along with image recovery, while the data hider cannot perform extraction.

A feasible method was first proposed in [8], in which the content owner encrypts an original image using a stream cipher, and the data hider embeds additional messages into ciphertext blocks by flipping three least significant bits (LSB) of half the pixels in each block. When extracting the additional messages, the recipient decrypts the marked encrypted image and generates two candidates for each block by flipping LSBs again. Since the original block is much smoother than the interfered, the embedded bits can be extracted and the original image perfectly recovered. This joint RDH-EI method depends on the size of each block. As long as the block size is appropriately chosen, errors of extraction and recovery can be avoided. This method was improved in [9] by exploiting spatial correlation between neighboring blocks and using a side-match algorithm to achieve a higher embedding rate. The flipping based approach was further improved in [10], in which multiple neighboring pixels in different locations are used to reduce error rates in extraction and recovery.

Recently, a new joint RDH-EI method was proposed in [11]. Data embedding is realized through a public key modulation mechanism. On the recipient end, a two-class SVM classifier is designed to distinguish encrypted and non-encrypted image patches. Consequently, the recipient can jointly extract the additional messages and recover the original image. This method provides a higher embedding capacity.

2.3 RDH-EI for JPEG Bitstream

As most RDH-EI methods are designed for uncompressed spatial-domain images, [12] proposes an approach capable of reversely hiding messages into encrypted JPEG bitstreams. This scheme aims at encrypting a JPEG bitstream into a properly organized structure and embedding additional messages into the encrypted bitstream by slight modifications. During the bitstream encryption, all appended bits of the Huffman codes are encrypted with a stream cipher, and all Huffman codes are kept unchanged. After encryption, the file size is preserved, and the format is compliant to common JPEG decoders. On the server side, the bitstream of every other block is selected as a candidate. If all AC coefficients of a candidate block are zero, the block is skipped. Additional bits are then encoded by LDPC-based error correction codes (ECC), and embedded into the useful candidate bitstream by flipping the LSBs of the encrypted appended bits of the AC coefficients in each candidate block. On the recipient side, LSBs of the appended bits of each candidate bitstream are flipped again to estimate the additional bits using a predefined blocking artifact function and an ECC decoder. Meanwhile, the original bitstream can be losslessly recovered according to the extracted bits.

In [27] and [28], some interesting ideas of RDH were proposed for JPEG images by combining image scrambling and data embedding. By scrambling the JPEG structure, additional message is embedded into the encrypted bitstream. However, in these methods data embedding must be combined with image encryption, which is different from general RDH-EI framework depicted in Fig. 2.

Limited by JPEG compression, large embedding capacity cannot be achieved. In [12], about 750 bits are embedded into the JPEG bitstream of a 512×512 grayscale image. The joint RDH-EI method requires a combined data extraction and image recovery. That may become a problem since the database administrator cannot read the hidden messages from the marked encrypted bitstream. As format compliance is required in JPEG encryption [29], it is difficult to design a secure encryption algorithm for JPEG. The algorithm previously presented in [12] is not secure enough. Analyses in [27] show that the principal structure of the original image can be estimated from the encrypted bitstream if all Huffman codes are kept unchanged. In view of these drawbacks, we provide a new encryption scheme for JPEG bitstream, and propose a separable RDH-EI approach for the encrypted bitstream. In the proposed method, data extraction and image recovery are separated, higher embedding capacity is achieved, and security of JPEG encryption enhanced.

3 Proposed Framework

The framework of the proposed method is depicted in Fig. 3. The JPEG RDH-EI workflow includes three parties: *content owner*, *data hider*, and *recipient*.

Given a JPEG bitstream and an encryption key, the content owner generates a ciphertext bitstream after syntax parsing and encryption. In the process, the file size is kept unchanged and the format is compliant to common JPEG decoders.

When a remote server receives the encrypted bitstream, the data hider parses the bitstream and hides additional messages in it using an embedding key. After the marked encrypted bitstream is constructed, the file size and format compliance are preserved. In this scheme, the server can extract additional messages from the marked encrypted bitstream using the embedding key.

On the recipient side, the additional messages can also be extracted from the received bitstream if the embedding key is available. A recipient with only the encryption key can view an approximate image by a direct decryption. If both the encryption and embedding keys are available, the recipient can losslessly recover the original bitstream after decrypting the marked encrypted JPEG bitstream.

3.1 JPEG Encryption and Decryption

In this section, we develop an encryption/decryption algorithm for baseline JPEG bitstreams. The encryption aims at preserving file size of the bitstream, avoiding leakage of image contents, and keeping the encrypted bitstream compliant to the common JPEG decoder. JPEG *compliance* here means an encrypted bitstream with suffix ".jpg" or ".jpeg" can be directly decoded by commonly-used JPEG decoders [29].

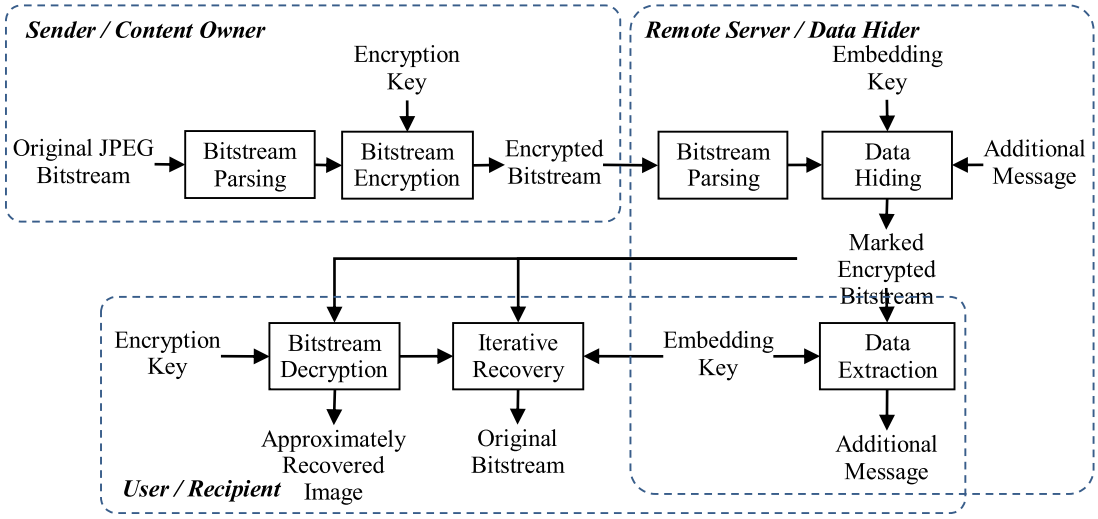

Fig. 3. Framework of the proposed method.

3.1.1 JPEG Encryption

Before encryption, the content owner parses the JPEG bitstream according to the simplified syntax of baseline [30], as shown in Fig. 4. We consider the syntax for compression of grayscale images. The JPEG format contains a *start-of-image* (**SOI**) marker, a JPEG header, the entropy encoded data, and an *end-of-image* (**EOI**) marker. The second layer from the top in Fig. 4 indicates that the entropy encoded data contains entropy-coded segments of all blocks. If a grayscale image sized $H \times W$ can be divided into N non-overlapping 8×8 blocks, there would be N entropy-coded segments, each corresponds to one block. The neighboring segments are separated by the *end-of-block* (**EOB**) markers. We denote each entropy-coded segment as $\mathbf{ECS}_i\, i = 1, 2, \ldots, N$.

Each entropy-coded segment contains codes of DC and AC coefficients, as shown in the third layer of Fig. 4. Denote the codes of DC and AC coefficients in the ith segment as $\mathbf{DCC}^{<i>}$ and $\mathbf{ACC}^{<i,j>}$, respectively, where $i = 1, 2, \ldots, N$ and $0 \leq j < 64$.

In the fourth layer, both codes of DC and AC coefficients contain Huffman code and appended bits. Let $\mathbf{DCH}^{<i>}$ and $\mathbf{DCA}^{<i>}$ be the Huffman codes and appended bits for DC coefficient, $\mathbf{ACH}^{<i,j>}$ and $\mathbf{ACA}^{<i,j>}$ for AC coefficient, respectively. Thus, each entropy-coded segment can be represented by

$$\begin{aligned}\mathbf{ECS}_i &= \{\mathbf{DCC}^{<i>}, \mathbf{ACC}^{<i,1>}, \mathbf{ACC}^{<i,2>}, \ldots, \mathbf{EOB}\} \\ &= \{\mathbf{DCH}^{<i>}, \mathbf{DCA}^{<i>}\}, \{\mathbf{ACH}^{<i,1>}, \mathbf{ACA}^{<i,1>}\}, \\ &\quad \{\mathbf{ACH}^{<i,2>}, \mathbf{ACA}^{<i,2>}\}, \ldots, \mathbf{EOB}\}.\end{aligned}$$

With an encryption key K_{enc}, the content owner pseudo-randomly selects entropy-coded segments corresponding to L blocks from the entropy encoded data, where $1 < L < N$. The encryption key K_{enc} is private to the content owner. Since all DC coefficients are encoded by DPCM starting from the first block, this block must be selected so that the encrypted bitstream can be correctly decoded by a JPEG decoder. Denote indexes of the *selected* L blocks as $\{S(1), S(2), \ldots, S(k), \ldots, S(L)\}$, and the *remaining* $N - L$ blocks as $\{R(1), R(2), \ldots, R(N-L)\}$. $S(\cdot)$ and $R(\cdot)$ are selection functions: $S(1) = 1, 1 < S(k) < N\,(k = 2, 3, \ldots, L)$, and $1 < R(i) < N\,(i = 1, 2, \ldots, N - L)$.

Next, the content owner generates a new bitstream including an **SOI** marker, a new JPEG header, entropy-

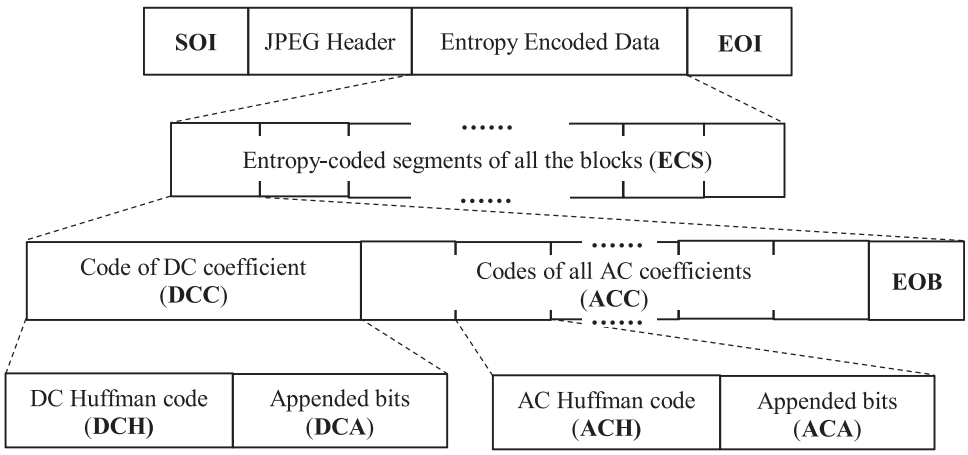

Fig. 4. Simplified syntax of JPEG baseline. **SOI**, **EOI** and **EOB** stand for start-of-image, end-of-image and end-of-block respectively. Acronyms in the parentheses are used in the discussion for brevity.

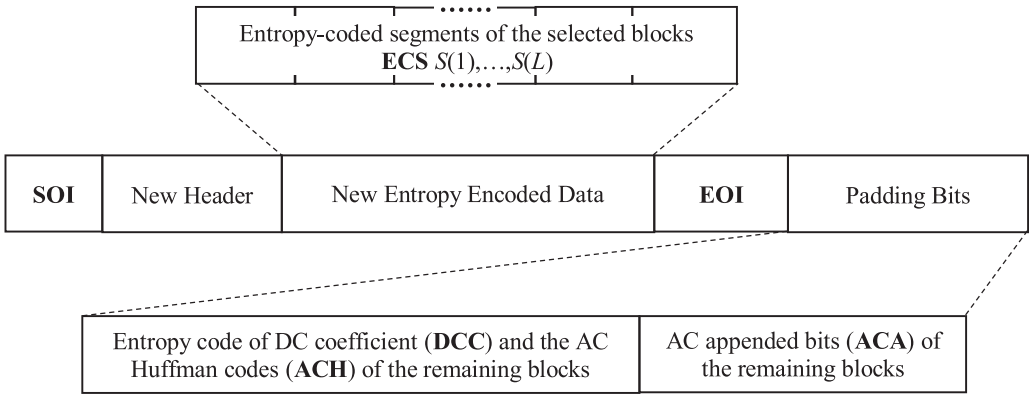

Fig. 5. Syntax of the new JPEG bitstream.

coded segments of the selected L blocks, an **EOI** marker, and *padding bits*. Two integers h and w, which are multiples of eight and satisfy $h \times w = 64L$, are chosen to specify the size of a new image. The new JPEG header is modified to store the size. Compressed bits of the remaining $N - L$ blocks are recomposed to construct the padding bits, whose syntax is illustrated in Fig. 5. These padding bits include two parts. The first part consists of the entropy code of DC coefficient and Huffman codes of all AC coefficients in the remaining blocks. The second part consists of the appended bits of all AC coefficients in the remaining blocks. We denote the padding bits as

$$\mathbf{P} = \{\mathbf{CC}, \mathbf{AP}\}$$

where

$$\begin{aligned}\mathbf{CC} = \{&\{\mathbf{DCC}^{<R(1)>}, \mathbf{ACH}^{<R(1),1>}, \mathbf{ACH}^{<R(1),2>}, \ldots, \mathbf{EOB}\}, \ldots,\\ &\{\mathbf{DCC}^{<R(N-L)>}, \mathbf{ACH}^{<R(N-L),1>},\\ &\mathbf{ACH}^{<R(N-L),2>}, \ldots, \mathbf{EOB}\}\}\end{aligned}$$

$$\begin{aligned}\mathbf{AP} = \{&\{\mathbf{ACA}^{<R(1),1>}, \mathbf{ACA}^{<R(1),2>}, \ldots\}, \ldots,\\ &\{\mathbf{ACA}^{<R(N-L),1>}, \mathbf{ACA}^{<R(N-L),2>}, \ldots\}\}\end{aligned}$$

As a result, when modifying **AP** to accommodate additional messages by a data hider, no Huffman codes are destroyed. This is why **ACH**s are separated from **ACA**s to make sure there is no Huffman code inside **AP**.

Assume there are M bits in the padding data and $\mathbf{P} = [p_1, p_2, \ldots, p_M]$. With the encryption key K_{enc} again, the content owner generates a key stream $\mathbf{K} = [k_1, k_2, \ldots, k_M]$ using a stream cipher algorithm such as RC4 and SEAL. The padding bits are then encrypted to $\mathbf{P}' = [p_1', p_2', \ldots, p_M']$ where

$$p_i' = p_i \oplus k_i, \quad 1 \le i \le M \tag{1}$$

In the same way, the content owner also encrypts all appended bits of the DC and AC Huffman codes inside the L selected segments.

Next, we embed the encrypted padding bits $\mathbf{P}'$ and the parameters H and W into the reserved application segments, marked as $\mathbf{APP_n}$ in the JPEG header, in the same way as [32]. After the processing, an encrypted JPEG bitstream is generated. The encrypted bitstream has the same amount of data as the original, and is compliant to the JPEG standard. As all bits between **SOI** and **EOI** are strictly structured following the JPEG syntax, the bitstream can be decoded to an image sized $h \times w$ using commonly-used JPEG decoders.

An example of the proposed JPEG encryption is shown in Fig. 6, in which (a) is a 512 × 512 image *Peppers* decoded from a plaintext JPEG bitstream, and (b) a 256 × 256 image decoded from an encrypted JPEG bitstream. In Fig. 6b, contents of the original image cannot be recognized for three reasons. First, the bitstream segments of L blocks are randomly selected from the original bitstream. Second, since the DC coefficients are represented by differential values, decoding DC codes in the selected segments gives results different from the original DC values. Third, as the appended bits of all coefficients are encrypted by a stream cipher, the decoded AC coefficients are different from the original values.

3.1.2 JPEG Decryption

When deciphering the encrypted bitstream, the new JPEG header and entropy encoded data can be extracted by parsing the bitstream. Meanwhile, the padding bits $\mathbf{P}' = [p_1', p_2', \ldots, p_M']$ can be extracted from the reserved application segments marked as $\mathbf{APP_n}$ in the JPEG header. With the encryption key K_{enc}, the appended bits of L selected entropy-coded segments and the padding bits $\mathbf{P} = \{\mathbf{CC}, \mathbf{AP}\}$ can be deciphered in the same way as (1).

(a) (b)

Fig. 6. Encryption of JPEG bitstream: (a) image decoded from the plaintext JPEG bitstream; (b) image decoded from the encrypted bitstream.

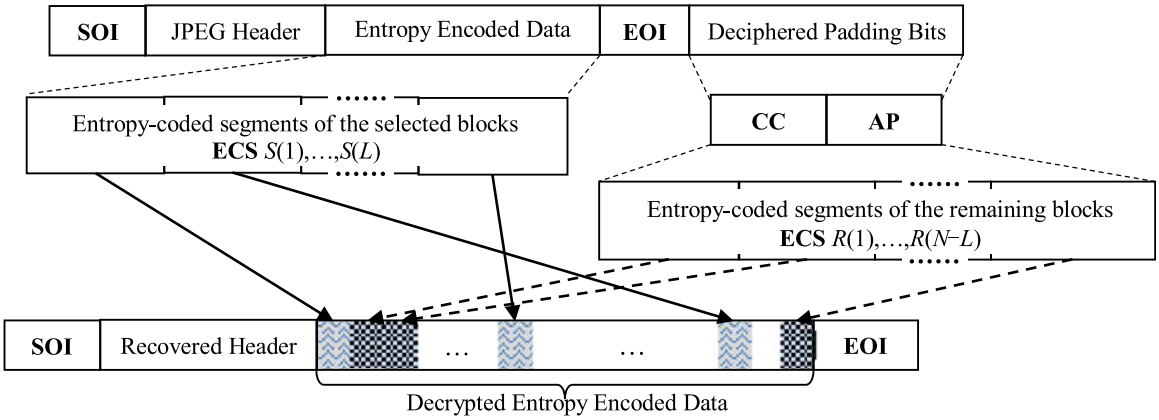

Fig. 7. Procedure of JPEG decryption.

From the JPEG header, we extract the DC and AC Huffman tables. With these tables, we parse the Huffman codes in **CC** and the appended bits in **AP** to reconstruct the $N-L$ remaining entropy-coded segments $\mathbf{ECS}_u$, $u = R(1), R(2), \ldots, R(N-L)$. Meanwhile, the L selected entropy-coded segments $\mathbf{ECS}_v\,(v = S(1), S(2), \ldots, S(L))$ are extracted from the new entropy encoded data. With the encryption key K_{enc}, the original indexes of the selected blocks can be recovered.

After that, the original JPEG bitstream is reconstructed containing **SOI**, the JPEG header, the decrypted entropy-coded segments, and **EOI**. The selected $\mathbf{ECS}_u\,(u = R(1), R(2), \ldots, R(N-L))$ and the remaining $\mathbf{ECS}_v\,(v = S(1), S(2), \ldots, S(L))$ are sequentially put back to the original positions, and the JPEG header is modified to restore the original image size $H \times W$. The decryption procedure is depicted in Fig. 7.

3.2 Data Hiding in Encrypted JPEG Bitstream

Based on the JPEG encryption algorithm, the content owner enciphers the JPEG bitstream and uploads the encrypted copy to a remote server. On the server side, the data hider extracts the encrypted padding bits from the header and embeds an additional message **M** into the encrypted padding bits. The procedure is depicted in Fig. 8. We denote all encrypted AC appended bits inside the encrypted padding bits as **A**, i.e., encrypted bits of **AP**, which contains m bits. Although it is difficult to identify the value of m directly from the encrypted padding bits, two solutions are provided at the end of this section.

The data hider evenly divides the binary vector **A** into s groups $\{\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_s\}$, where $s = m/n$, $n = \beta{\cdot}e$, β is a positive integer, and e is the average number of appended bits of all AC coefficients inside each block. The value of e is identified by parsing all AC appended bits in L selected entropy-coded segments. Assuming there are m_a such bits, the value of e is calculated by $e = m_a/L$.

Subsequently, the data hider constructs a $k \times n$ binary matrix **H** by

$$\mathbf{H} = [\mathbf{I}_{k\times k}, \mathbf{Q}_{k\times r}] \tag{2}$$

where **Q** is a pseudo-randomly generated binary matrix, and $r = n - k$. Many algorithms like RC4 and SEAL can be used to generate **Q**.

For each group $\mathbf{A}_t\,(t = 1, 2, \ldots, s)$, the data hider further computes

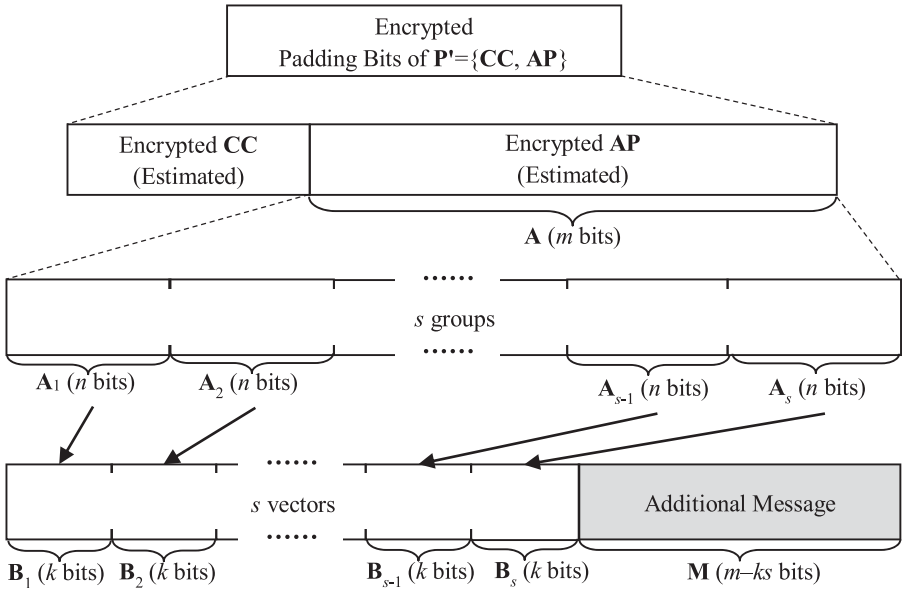

Fig. 8. Procedure of data embedding.

$$[\mathbf{B}_t(1), \mathbf{B}_t(2), ..., \mathbf{B}_t(k)]^T = \mathbf{H} \cdot [\mathbf{A}_t(1), \mathbf{A}_t(2), ..., \mathbf{A}_t(n)]^T \quad (3)$$

in which all calculations are binary arithmetic. Thus, each group $\mathbf{A}_t$ containing n bits is compressed to a vector $\mathbf{B}_i$ with k bits. After compressing all groups from m bits to ks bits, the additional message $\mathbf{M}$ containing $m - ks$ bits are appended to generate

$$\mathbf{P}_m = \{\mathbf{CC}, \{\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_s, \mathbf{M}\}\}.$$

The structure is depicted in Fig. 8.

With an embedding key K_{emb}, the data hider shuffles $\mathbf{P}_m$ to produce a sequence $\mathbf{E}$. These bits are then embedded into the reserved application segments marked as $\mathbf{APP_n}$ in JPEG header. This way, a *marked encrypted JPEG bitstream* containing additional messages is generated.

To extract the additional messages, one can reshuffle the padding bits of the marked encrypted bitstream using the embedding key K_{emb}, and obtain the additional messages by extracting the last $m - ks$ bits. Since extraction is done in the encrypted domain without revealing image contents, the proposed RDH-EI method is therefore separated from image recovery.

Next, we explain how the parameter m is estimated. There are two possible solutions. A simple solution is to transmit the parameter m along with the encrypted JPEG bitstream, by embedding it inside the reserved application segments marked as $\mathbf{APP_n}$ in JPEG header. Another is to estimate the length of encrypted padding bits. If there are n_s bits in L selected entropy-coded segments and n_r bits in the $N - L$ remaining segments, m can be estimated by $m = \lfloor \lambda \cdot (n_r/n_s) \cdot m_a \rfloor$, where $\lfloor \cdot \rfloor$ is a rounding operator, and $\boldsymbol{\lambda}$ a scaling factor ($0 < \lambda < 1$) used to avoid modifying the encrypted Huffman codes during data hiding.

3.3 Iterative Recovery of Original Image

On the recipient side, the marked encrypted JPEG bitstream can be directly decoded by the JPEG decoder to construct an encrypted image of a smaller size. With the embedding key K_{enc}, the recipient can parse and decipher the marked encrypted JPEG bitstream using the proposed JPEG decryption algorithm described in Section 3.1. Since only the AC appended bits of the remaining entropy-coded segments were modified in data hiding, an approximate image with reduced quality can be reconstructed after decryption.

With both the encryption and embedding keys, the recipient can losslessly recover the original JPEG image. Although the compression algorithm in (3) is irreversible, we have several solutions to identify the original bits according to the changes of blocking artifacts. A flowchart of the recovery is shown in Fig. 9.

The recipient first parses and extracts the encrypted padding bits $\mathbf{E}$ from the marked encrypted JPEG bitstream and reshuffles $\mathbf{E}$ to restore $\mathbf{P}_m$ using the embedding key K_{emb}, where

$$\mathbf{P}_m = \{\mathbf{CC}, \{\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_s, \mathbf{M}\}\}.$$

According to the binary matrix $\mathbf{H}$, the recipient generates a binary matrix $\mathbf{G}$,

$$\mathbf{G} = [\mathbf{Q}_{k\times r}^T, \mathbf{I}_{r\times r}] \quad (4)$$

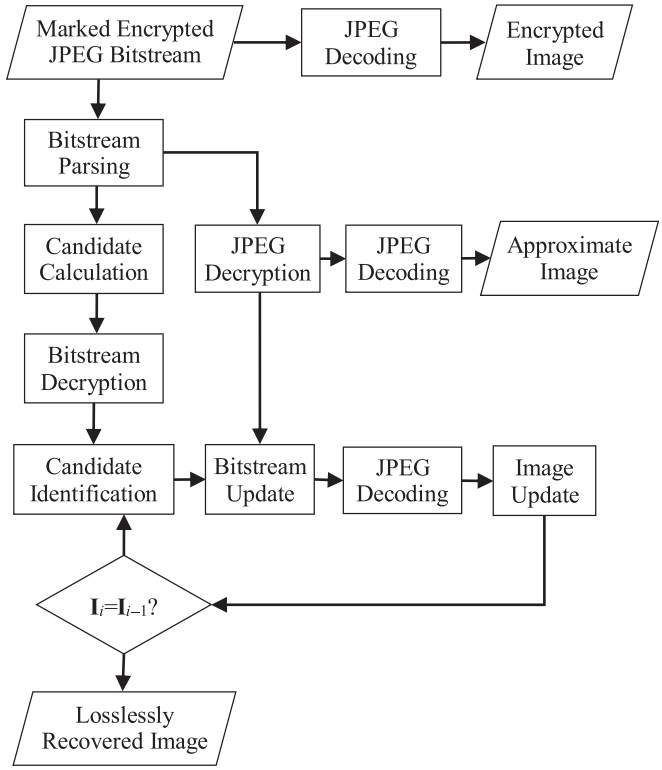

Fig. 9. Flow chart of iterative recovery.

where $r = n - k$ and $n = \beta e$. For each group $\mathbf{B}_t$, the recipient calculates

$$\begin{aligned}&\left[\mathbf{A}_t{}^{(c)}(1), \mathbf{A}_t{}^{(c)}(2), ..., \mathbf{A}_t{}^{(c)}(n)\right] \\ &= [\mathbf{B}_t(1), \mathbf{B}_t(2), ..., \mathbf{B}_t(k), 0, ...0] + [a_1, a_2, ..., a_r] \cdot \mathbf{G}\end{aligned} \quad (5)$$

where $[a_1, a_2, \ldots, a_r]$ is an arbitrary binary vector, and $\mathbf{A}_i^{(c)}$ the 2^r possible candidates for each group $\mathbf{A}_t$, $t = 1, 2, \ldots, s$, and $c = 1, 2, \ldots, 2^r$.

With the encryption key K_{enc}, the candidate vectors $\mathbf{A}_t^{(c)}$ are decrypted to plaintext candidates $\mathbf{D}_t^{(c)}$ using the stream cipher algorithm. According to the syntax of the recomposed JPEG bitstream, lossless recovery is equivalent to identifying the suitable one from the 2^r possible plaintext candidates to recover the padding bits.

Assume that the AC appended bits in each candidate $\mathbf{D}_t^{(c)}$ belongs to several entropy-encoded segments, i.e.,

$$\mathbf{D}_t^{(c)} = \{\mathbf{F}, \underline{\mathbf{ACAP}}^{<t1>}, \underline{\mathbf{ACAP}}^{<t2>}, \ldots, \underline{\mathbf{ACAP}}^{<tl>}, \mathbf{F}\}.$$

Here $\mathbf{F}$ stands for the fragment AC appended bits of a segment, and $\underline{\mathbf{ACAP}}^{<ta>}$ for the candidate of all AC appended bits in the t_ath remaining segment, $t = 1, 2, \ldots, s$, $t_a \in R(1), R(2), \ldots, R(N-L)$, and $a = 1, 2, \ldots, l$. The value of l is identified by parsing the AC Huffman codes in the decrypted padding bits. Meanwhile,

$$\underline{\mathbf{ACAP}}^{<ta>} = \{\underline{\mathbf{ACA}}^{<ta,1>}, \underline{\mathbf{ACA}}^{<ta,2>}, \ldots\}$$

where $\underline{\mathbf{ACA}}^{<ta,j>}$ is a candidate of appended bits, and $0 \le j < 64$.

Thus, 2^r candidates for l entropy-coded segments can be generated,

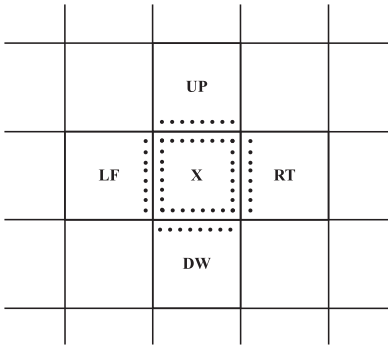

Fig. 10. Blocking artifacts.

$$\underline{\mathbf{ECS}}_{ta}^{(c)} = \{\mathbf{DCC}^{<ta>}, \mathbf{ACH}^{<ta,1>}, \underline{\mathbf{ACA}}^{<ta,2>}, \ldots, \mathbf{EOB}\}$$

Accordingly, 2^r candidate pixel blocks $\{\mathbf{PB}_{t1}^{(c)}, \mathbf{PB}_{t2}^{(c)}, \ldots, \mathbf{PB}_{tl}^{(c)}\}$ are constructed by entropy decoding $\{\underline{\mathbf{ECS}}_{t1}^{(c)}, \underline{\mathbf{ECS}}_{t2}^{(c)}, \ldots, \underline{\mathbf{ECS}}_{tl}^{(c)}\}$.

Next, the recipient identifies the suitable candidate $\mathbf{D}_t^{(bt)}$ from $\mathbf{D}_t^{(c)}$ ($c = 1, 2, \ldots, 2^r$) by calculating the blocking artifacts of the candidate blocks, where

$$b_t = \arg\min_{c} \sum_{a=t_1}^{t_l} f(\mathbf{PB}_a{}^{(c)}) \tag{6}$$

and f is the blocking artifact function,

$$f(\mathbf{X}) = \sum_{i=1}^{8} |\mathbf{X}(1,i) - \mathbf{UP}(8,i)| + |\mathbf{X}(i,1) - \mathbf{LF}(i,8)|. \tag{7}$$

The blocking artifact function is illustrated in Fig. 10, in which **X** is the present block, **UP** and **LF** are the up and left neighboring blocks. We use the up and left blocks when calculating blocking artifact as blocks are constructed orderly from left-to-right and top-to-bottom, and only these two blocks are available in the first round of recovery.

The reason we use this function as the criteria of recovery is that data embedding by coefficient modification always increases blocking artifacts [12]. It should be noted that over-smooth in the recovered image may occur when a small parameter n is used during embedding. As long as n is large, i.e., enough blocks are used together to evaluate the blocking artifacts, the over-smooth effect can be avoided.

After sequentially processing all groups $\{\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_s\}$, suitable candidates $\{\mathbf{D}_1{}^{(b1)}, \mathbf{D}_2{}^{(b2)}, \ldots, \mathbf{D}_s{}^{(bs)}\}$ constituting updated padding bits are identified. This way, a new image $\mathbf{I}_0$ is constructed by decoding the updated JPEG bitstream. The image $\mathbf{I}_0$ has better quality than the directly deciphered image.

The first round of recovery using blocking artifact function in (7) may be inaccurate. In the next stage, the recipient iteratively refines the updated JPEG bitstream to losslessly recover the original image, using contents in the existing image as a reference. With the updated padding bits and the candidates $\mathbf{D}_t^{(c)}$ ($c = 1, 2, \ldots, 2^r$), the recipient iteratively finds the best candidates using (6) and (8), in which the blocking artifact function f is different from (7).

$$\begin{aligned} f(\mathbf{X}) = \sum_{i=1}^{8} &|\mathbf{X}(1,i) - \mathbf{UP}(8,i)| + |\mathbf{X}(i,1) - \mathbf{LF}(i,8)| \\ &+ |\mathbf{X}(8,i) - \mathbf{DW}(1,i)| + |\mathbf{X}(i,8) - \mathbf{RT}(i,1)| \end{aligned} \tag{8}$$

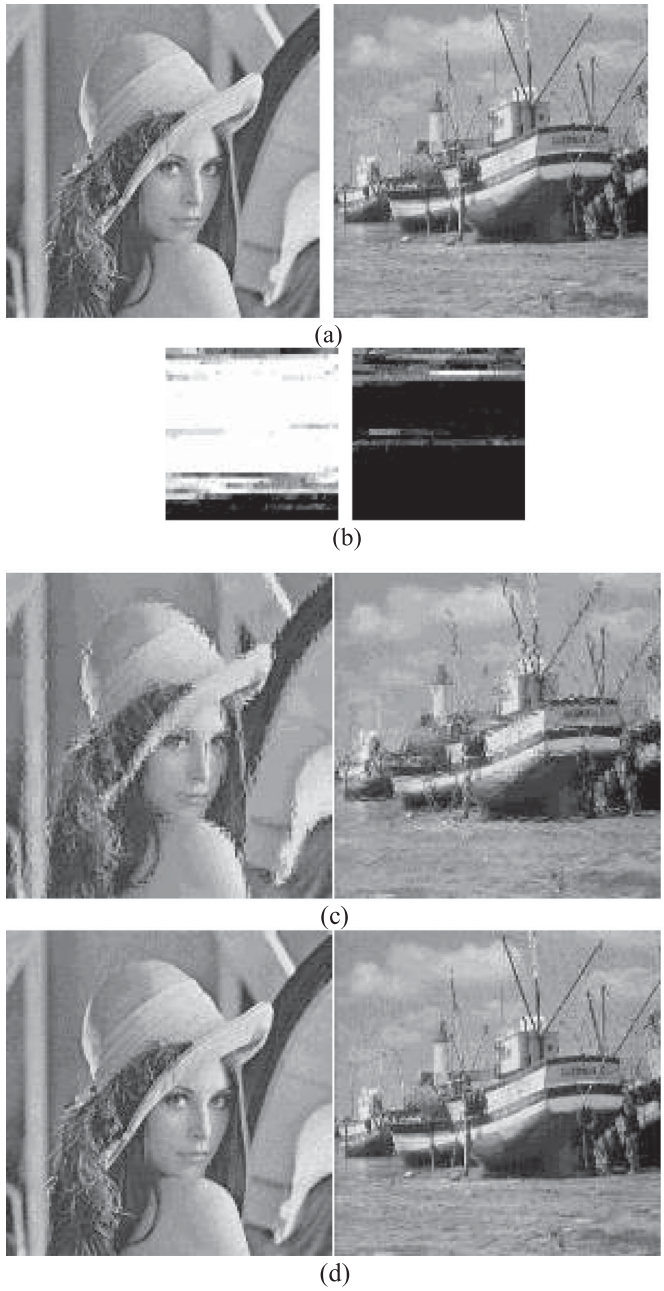

Fig. 11. RDH-EI in the bitstreams corresponding to *Lena* and *Boat*: (a) original JPEG images, (b) encrypted images, (c) directly decrypted images, and (d) losslessly recovered images.

As shown in Fig. 10, **UP**, **LF**, **DW** and **RT** are neighboring blocks around the present block **X**.

In each iterative step, the recipient updates the bitstream and generates a refined image. The generated image is compared with the resulting image of the previous round. The iteration runs until no difference is found. This way, the recipient can losslessly recover the original JPEG image.

4 Experimental Results and Analysis

To verify the proposed method, we use a set of grayscale images sized 512 × 512, and compress them to JPEG bitstreams with different quality factors. The encrypted padding bits, and the parameters m, H, and W are hidden into the JPEG header. Let $\alpha = L/N$ be the ratio of selected blocks.

TABLE 1
PSNR (dB) and Payload C_e (bits) of *Lena*

r	β			
	9	10	11	12
1	$+\infty$, 341	$+\infty$, 308	$+\infty$, 279	$+\infty$, 259
2	$+\infty$, 682	$+\infty$, 616	$+\infty$, 558	$+\infty$, 518
3	$+\infty$, 1,023	53.6, 924	$+\infty$, 837	$+\infty$, 777
4	$+\infty$, 1,364	$+\infty$, 1,232	$+\infty$, 1,116	$+\infty$, 1,036

TABLE 2
PSNR (dB) and Payload C_e (Bits) of *Baboon*

r	β			
	9	10	11	12
1	43.1, 341	38.2, 307	$+\infty$, 279	$+\infty$, 256
2	$+\infty$, 682	$+\infty$, 614	$+\infty$, 558	$+\infty$, 512
3	40.2, 1,023	42, 921	38.3, 837	$+\infty$, 768
4	35.1, 1,364	31.6, 1,228	33.3, 1,116	34.7, 1,024

TABLE 3
PSNR (dB) and Payload C_e (Bits) *Texmos*

r	β			
	9	10	11	12
1	$+\infty$, 341	$+\infty$, 307	$+\infty$, 279	$+\infty$, 256
2	37.8, 682	$+\infty$, 614	$+\infty$, 558	$+\infty$, 512
3	32.5, 1,023	35.4, 921	$+\infty$, 837	$+\infty$, 768
4	34.4, 1,364	31.3, 1,228	$+\infty$, 1,116	$+\infty$, 1,024

An example is given in Fig. 11, in which (a) shows the original images *Lena* and *Boat* compressed with a quality factor 80. After encrypting the bitstreams with an encryption key K_{enc} and the parameter $\alpha = 0.25$, format-compliant bitstreams are generated. The encrypted bitstreams can be decoded to smaller images sized 256 $\times$ 256 by a JPEG decoder, which are shown in Fig. 11b. The encrypted bitstreams have the same lengths as the original. Secret messages containing 1,023 bits are embedded into each encrypted bitstream using an embedding key K_{emb} and parameters $\beta = 9$ and $r = 3$. After data hiding, the marked encrypted JPEG bitstreams can still be directly decoded to images by JPEG decoder. On the recipient side, additional messages can be extracted without any errors if the key K_{emb} is available. Approximate images can be generated by decrypting the encrypted bitstream containing additional message using the key K_{enc}. Fig. 11c shows the approximate images, with PSNR being 21.7 and 20.4 dB, respectively. When both keys are available, the original bitstreams can be recovered without loss. Fig. 11d shows the losslessly recovered images after three iterations.

In Tables 1, 2, and 3, we show the embedding payloads C_e and PSNR of the recovered images, using images *Lena*, *Baboon* and *Texmos*. The PSNR value $+\infty$ means lossless recovery. We use a fixed ratio $\alpha = 0.25$, i.e., messages with 1,540 bits are embedded into the encrypted bitstream of *Lena* with $\beta = 10$ and $r = 5$. The experimental results show that the original images can be losslessly recovered if β is not too small and r is not too large. The image *Texmos* is shown in Fig. 12, indicating that the proposed method is also applicable to natural images with rich textures.

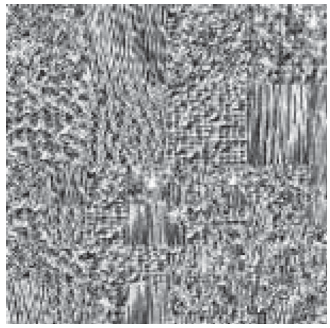

Fig. 12. The image *Texmos*.

TABLE 4
Empirical Parameters to Achieve Lossless Recovery

r	1	2	3	4	5
β	15	18	22	20	22

TABLE 5
Comparison of Payloads (Bits)

Images	[8]	[9]	[12]	Proposed
Lena	1,024	1,024	750	1,364
Baboon	256	334	750	768
Man	655	1,024	750	1,368
Sailboat	655	1,024	750	1,364

TABLE 6
Payloads Corresponding to Different Selection Ratio α

α	0.0005	0.042	0.083	0.125	0.167	0.208	0.250
Lena	1,026	981	939	897	855	810	768
Peppers	1,026	981	939	897	855	810	768
Airplane	1,023	984	939	897	852	810	768
Man	1,023	981	939	897	855	810	768
Boats	1,023	981	936	894	852	810	768

In real applications, parameters β and r can be chosen empirically in the same way as the previous RDH-EI works [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19]. For different types of images, some images are used for training. For example, we arbitrarily choose 100 natural images sized 512 $\times$ 512, and perform embedding using the proposed method. Table 4 shows the used empirical parameters, with which the original JPEG image can be losslessly recovered.

To find the achievable payload, we compare the proposed methods with [8] and [9] designed for uncompressed images. Experimental results listed in Table 5 show that payloads of the proposed method for JPEG are close to [8] and [9]. We mainly compare the proposed method with [12] that is also an RDH-EI for encrypted JPEG bitstream. Results given in Table 5 indicate that the proposed method has much larger achievable payloads than that of [12]. We arbitrarily choose 50 JPEG images to embed 750 bits into each one using both methods. Average PSNR obtained with the proposed method is about 7 dB smaller than [12]. Although quality of the directly decrypted image in [12] is better than the proposed method, better PSNR in [12] is achieved by sacrificing security.

Since α is the ratio of the selected blocks, a smaller α leads to more padding bits used to carry more secret message. Table 6 shows embedding payload corresponding to

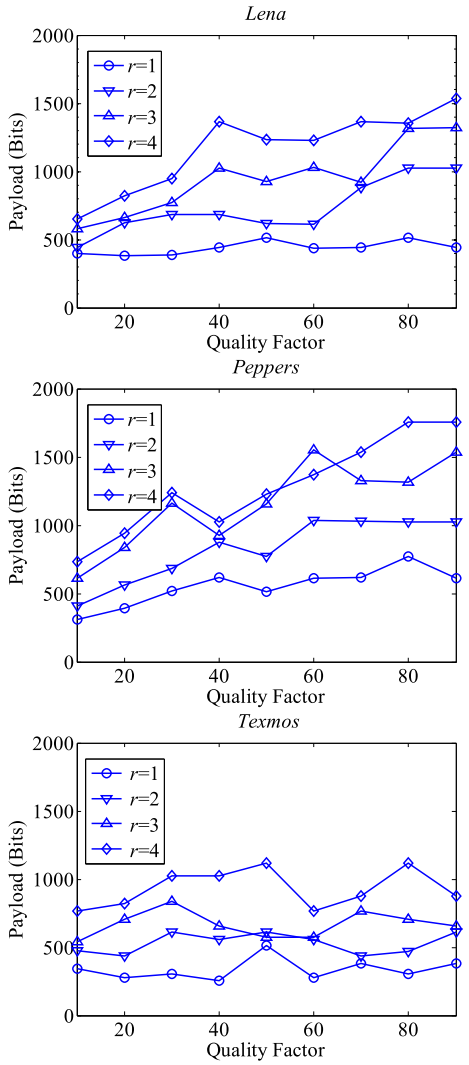

Fig. 13. Payload *versus* quality factor.

different values of α, in which fixed parameter values $r = 3$ and $\beta = 12$ are used to ensure lossless recovery. Experimental results indicate that a larger embedding payload can be achieved with a smaller α.

Quality factor used for scaling the default quantization table is an important factor in JPEG compression, and it also impacts embedding payloads. We use the popular tool IJG [31] to scale the quantization table as suggested in the JPEG standard [30]. In the standard, the quality factor ranges from 0 to 100, and the default is 50. We use $\alpha = 0.25$, different values of r, and corresponding values of β to achieve lossless recovery. These parameters correspond to different payloads. Fig. 13 shows relations between the payload and the quality factor ranging from 10 to 90, based on *Lena*, *Peppers* and *Texmos*. Generally, higher payloads can be achieved with larger quality factors or larger values of r.

To check the convergence property of the iterative recovery, we conduct a set of experiments. A total of 1,000 arbitrary images in Bossbase1.01 [33] are used, with $r = 1$, $\alpha = 0.25$, different β, and different quality factors Q. Histograms in Fig. 14 show that, in all experiments, the required number of iterative rounds is no more than 5. Table 7 shows the average convergence speed of 1,000 images using different parameters. In general, several hundreds of seconds are needed to recover each image. When β gets smaller, longer time is required.

We also evaluate relations between embedding payloads and image resolutions. Fixed parameters $r = 3$ and $\beta = 12$ are used to implement the proposed method to achieve lossless recovery. We use a parameter S to represent the image resolution, and let the image size be $2^{S/2} \times 2^{S/2}$. Fig. 15 shows that the achievable payload increases as the image gets larger. Resolution of an original JPEG image should be larger than 16×16. Otherwise, few messages can be embedded into the encrypted image.

The proposed system is securer than [12]. In [12], the quantization tables and all appended bits are encrypted,

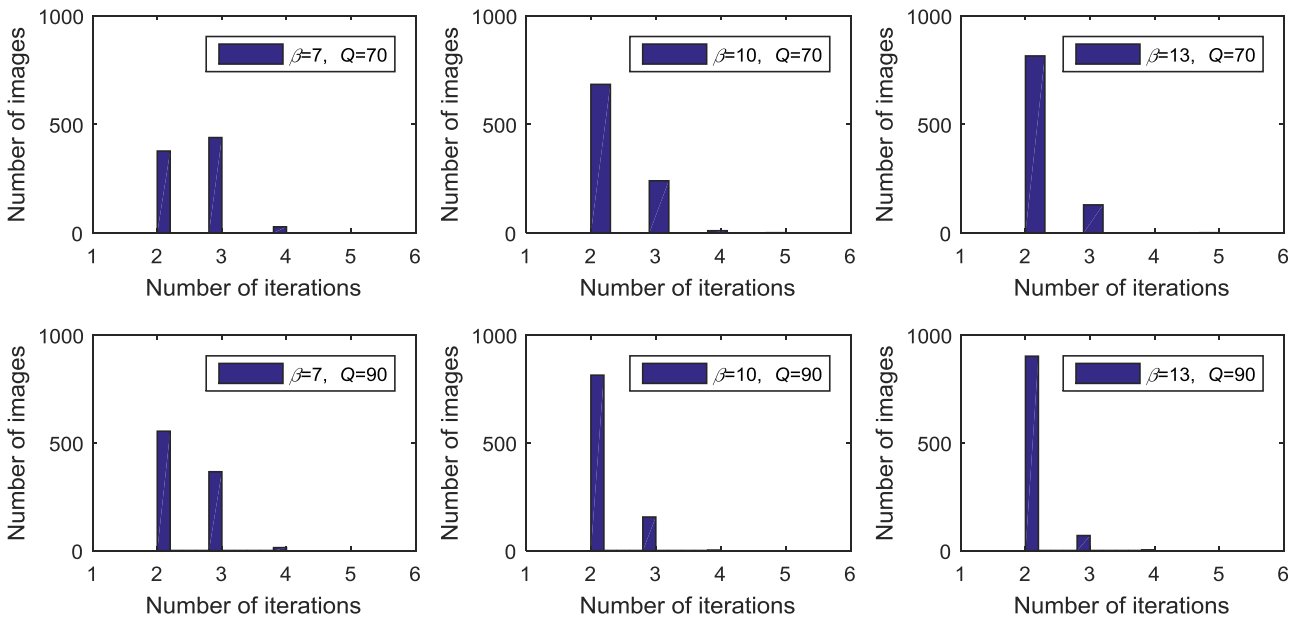

Fig. 14. Convergence of iterative recovery.

TABLE 7
Average Convergence Speed Corresponding to Different Parameters (in Seconds)

Q	β		
	7	10	13
70	432.5	365.6	338.4
90	466.8	377.9	236.7

while all Huffman codes are unchanged. If an adversary uses all Huffman codes and sets all appended bits to zero, a contour of the original image can be revealed. While in the proposed method, only part of the bitstream segments are randomly selected to construct a smaller sized JPEG image. If α is small, most of the entropy-coded segments are rearranged to the padding bits that are further encrypted with a stream cipher. Thus, the adversary cannot reconstruct a contour of the original image with limited Huffman codes. As an example, we use [12] and the proposed method to encrypt the original JPEG images *Baboon* and *Peppers*, both sized 512 × 512. The encrypted images constructed by [12] are sized 512 × 512, while the encrypted images by the proposed method are sized 256 × 256. Fig. 16a shows revealed contours by using all Huffman codes in the encrypted bitstream of [12]. In Fig. 16b, no contours are revealed, indicating that the proposed method is much securer than [12].

Besides, the proposed method is also secure against ciphertext-only attacks. Two parts are included in the proposed encryption, i.e., block selection and stream cipher. L entropy-coded segments are pseudo-randomly selected from the JPEG bitstream, in which the first entropy-coded segment must be selected. After permuting the $L-1$ segments, a new entropy encoded bitstream is constructed. When L is large enough, it is difficult for an adversary to find the original orders from $C_{N-1}^{L-1}\cdot(L-1)!$ possibilities. On the other hand, all appended bits in the L selected segments and all bits in the remaining $N-L$ segments are encrypted by stream cipher. Thus, an adversary is unable to break the original bits and reconstruct the original image as long as the encryption key is long enough, e.g., 128 bits for RC4. We create images sized 256 × 256, 512 × 512 and 1,024 × 1,024 by sampling or interpolating the original *Lena*. After JPEG compressing these images using a quality factor 80, lengths of the compressed bits of these images are 57,074, 213,435 and 435,620 bits respectively. The results

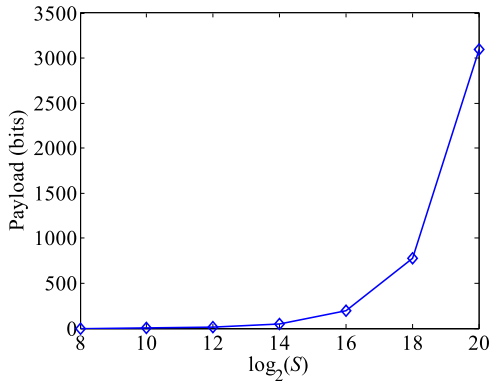

Fig. 15. Payload *versus* image resolution.

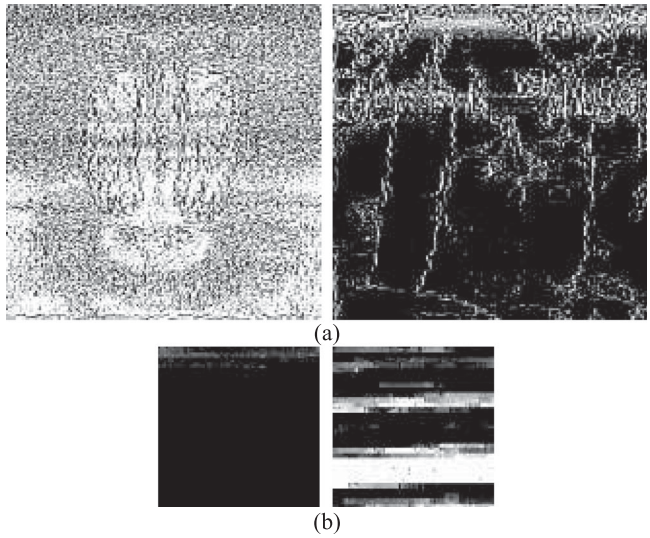

Fig. 16. Security analyses using encrypted bitstreams of *Baboon* and *Peppers*: (a) constructed 512 × 512 images by attacking [12], and (b) constructed 256 × 256 images by attacking the proposed method.

show that it is difficult for an adversary to break so many bits using the brute-force attack.

Denote average complexity of deciphering and decoding the bitstream of each block as T_c, that of bitstream decoding for each block as T_b, and that of blocking artifact calculation as T_a. We compare recovery complexities of the proposed method with that of [12]. In [12], the procedure of recovery includes bitstream deciphering, bitstream decoding, and blocking artifact calculation. As a result, average complexity of recovering one block is approximately $T_c+T_b+T_a$. In the proposed method, 2^r possible candidates are first calculated using (4). For each solution, bitstream deciphering, bitstream decoding, and blocking artifact calculation are used to find the best solution. After that, an iterative algorithm is used to complete the recovery, with a procedure close to the first round. As a result, average complexity of recovering one block is approximately $2^r\cdot\tau\cdot(T_c+T_b+T_a)$, where τ is the number of iterative rounds. Therefore, complexity of the proposed method for recovering one block is $2^r\cdot\tau$ times that of [12]. Since $r\geq 1$ and $\tau\geq 1$, complexity of the proposed method is higher than that of the previous work. This is the cost paid for separated operations, higher payload and better security.

5 Conclusion

This paper proposes a separable reversible data hiding scheme for the encrypted JPEG bitstream. A JPEG encryption and decryption algorithm is developed to hide the content of an original image. When the server receives the enciphered bitstream, a data hider can embed additional messages into the encrypted copy by compressing the padding bits of the bitstream. With an iterative recovery method based on blocking artifacts, the recipient can losslessly recover the original bitstream. The proposed method provides larger embedding capacity than the previous approach. It is separable because anyone who has the embedding key can extract the additional message from the marked encrypted bitstream without revealing the original content of the JPEG image.

The proposed method also offers better security than the previous work. A new JPEG bitstream corresponding to a smaller sized image is constructed. Therefore, information leakage of the original content, e.g., contour of an image, can be avoided. The procedure is realized by rearranging some entropy-coded segments to generate the padding bits. These bits are embedded into the reserved segments labeled by $\mathbf{APP_n}$ in the JPEG header. The encrypted bitstream can still be decoded by the commonly-used decoders, e.g., the decoder incorporated in the Windows operating system. Meanwhile, the amount of data of the bitstream is unchanged.

Acknowledgments

This work was supported by Natural Science Foundation of China (Grant 61572308, Grant U1536108, Grant 61525203, Grant 61572452, and Grant 61472235). The authors thank Professor Shuozhong Wang for his assistance in improving the quality of this paper. Zhenxing Qian is the corresponding author.

References

[1] Z. Erkin, et al., "Protection and retrieval of encrypted multimedia content: When cryptography meets signal processing," *EURASIP J. Inf. Secur.*, vol. 2007, 2007, Art. no. 17.
[2] M. Barni, T. Kalker, and S. Katzenbeisser, "Inspiring new research in the field of signal processing in the encrypted Domain," *IEEE Signal Process. Mag.*, vol. 30, no. 2, p. 16, 2013.
[3] W. Liu, W. Zeng, L. Dong, and Q. Yao, "Efficient compression of encrypted grayscale images," *IEEE Trans. Image Process.*, vol. 19, no. 4, pp. 1097–1102, Apr. 2010.
[4] 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.
[5] Y. Rahulamathavan, R. Phan, J. Chambers, and D. Parish, "Facial expression recognition in the encrypted domain based on local fisher discriminant analysis," *IEEE Trans. Affective Comput.*, vol. 4, no. 1, pp. 83–92, Jan. 2013.
[6] T. Bianchi, and A. Piva, "TTP-Free asymmetric fingerprinting based on client side embedding," *IEEE Trans. Inf. Forensics Secur.*, vol. 9, no. 10, pp. 1557–1568, Oct. 2014.
[7] Z. Fu, X. Sun, Q. Liu, L. Zhou, and J. Shu, "Achieving efficient cloud search services: Multi-keyword ranked search over encrypted cloud data supporting parallel computing," *IEICE Trans. Commun.*, vol. 98, no. 1, pp. 190–200, 2015.
[8] X. Zhang, "Reversible data hiding in encrypted images," *IEEE Signal Process. Lett.*, vol. 18, no. 4, pp. 255–258, Apr. 2011.
[9] W. Hong, T. Chen, and H. Wu, "An improved reversible data hiding in encrypted images using side match," *IEEE Signal Process. Lett.*, vol. 19, no. 4, pp. 199–202, Apr. 2012.
[10] X. Liao and C. Shu, "Reversible data hiding in encrypted images based on absolute mean difference of multiple neighboring pixels," *J. Vis. Commun. Image Representation*, vol. 28, pp. 21–27, 2015.
[11] 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.
[12] 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.
[13] X. Zhang, "Separable reversible data hiding in encrypted image," *IEEE Trans. Inf. Forensics Secur.*, vol. 7, no. 2, pp. 826–832, 2012.
[14] X. Wu, and W. Sun, "High-capacity reversible data hiding in encrypted images by prediction error," *Signal Process.*, vol. 104, pp. 387–400, 2014.
[15] Z. Qian, and X. Zhang, "Reversible data hiding in encrypted image with distributed source encoding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 4, pp. 636–646, Apr. 2016.
[16] K. Ma, et al., "Reversible data hiding in encrypted images by reserving room before encryption," *IEEE Trans. Inf. Forensics Secur.*, vol. 8, no. 3, pp. 553–562, Mar. 2013.
[17] W. Zhang, K. Ma and N. Yu, "Reversibility improved data hiding in encrypted images," *Signal Process.*, vol. 94, pp. 118–127, 2014.
[18] 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.
[19] M. Fujiyoshi, "Separable reversible data hiding in encrypted images with histogram permutation," in *Proc. IEEE Int. Conf. Multimedia Expo*, 2013, pp. 1–4.
[20] Z. Qian, X. Zhang, and G. Feng, "Reversible data hiding in encrypted images based on progressive recovery," *IEEE Signal Process. Lett.*, vol. 23, no. 11, pp. 1672–1676, Nov. 2016.
[21] Z. Qian, X. Zhang, Y. Ren, and G. Feng, "Block cipher based separable reversible data hiding in encrypted images," *Multimedia Tools Appl.*, vol. 75, no. 21, pp. 13749–13763, 2016
[22] Z. Qian, S. Dai, F. Jiang, and X. Zhang, "Improved joint reversible data hiding in encrypted images," *J. Vis. Commun. Image Representation*, vol. 40, pp. 732–738, 2016
[23] T. Kalker and F. M. Willems, "Capacity bounds and code constructions for reversible data-hiding," in *Proc. 14th Int. Conf. Dig. Signal Process.*, 2002, pp. 71–76.
[24] M. Cancellaro, F. Battisti, M. Carli, G. Boato, F. G. B. De Natale, and A. Neri, "A commutative digital image watermarking and encryption method in the tree structured Haar transform domain," *Signal Process. Image Commun.*, vol. 26, pp. 1–12, 2011.
[25] S. Lian, Z. Liu, Z. Ren, and H. Wang, "Commutative encryption and watermarking in video compression," IEEE Trans. Circuits and Systems for Video Technology, vol. 17, no. 6, pp. 774–778, 2007.
[26] X. Zhang, "Commutative reversible data hiding and encryption," *Secur. Commun. Netw.*, vol. 6, no. 11, pp. 1396–1403, 2013.
[27] S. Y. Ong, K. S. Wong, X. Qi, and K. Tanaka, "Beyond format-compliant encryption for JPEG image," *Signal Process.: Image Commun.*, vol. 31, pp. 47–60, 2015.
[28] S. Y. Ong, K. S. Wong, and K. Tanaka, "Scrambling-embedding for JPEG compressed image," *Signal Process.*, vol. 109, pp. 56–68, 2015.
[29] A. Massoudi, F. Lefebvre, C. De Vleeschouwer, B. Macq, and J.-J. Quisquater, "Overview on selective encryption of image and video: Challenges and perspectives," *EURASIP J. Inf. Secur.*, vol. 5, pp. 1–18, 2008.
[30] Int. Tele. Union, CCITT Recommendation T.81, "Information technology-digital compression and coding of continuous-tone Still Images - Requirements and Guidelines," 1992.
[31] Independent JPEG Group. (2016, Jan.). [Online]. Available: http://www.ijg.org/
[32] T. Richter, A. Artusi, and T. Ebrahimi, "JPEG XT: A new family of JPEG backward-compatible standards," *IEEE Multimedia*, vol. 23, no. 3, pp. 80–88, Jul.-Sep. 2016.
[33] Bossbase Database. (2013). [Online]. Available: http://dde.binghamton.edu/download/

Zhenxing Qian (M'12) received the BS and the PhD degrees both from University of Science and Technology of China, in 2003 and 2007, respectively. He is currently a professor in the School of Communication and Information Engineering, Shanghai University, China. His research interests include data hiding and multimedia security. He is a member of the IEEE.

Hang Zhou received the BS degree from the School of Communication and Information Engineering, Shanghai University. He is currently working toward the master's degree at the University of Science and Technology of China. His research interests include information hiding and image processing.

Xinpeng Zhang (M'11) received the BS degree in computational mathematics from Jilin University, China, in 1995, and the ME and PhD degrees in communication and information system from Shanghai University, China, in 2001 and 2004, respectively. Since 2004, he has been with the faculty of the School of Communication and Information Engineering, Shanghai University, where he is currently a professor. His research interests include information hiding, image processing and digital forensics. He is a member of the IEEE.

Weiming Zhang received the MS and PhD degrees from Zhengzhou Information Science and Technology Institute, China, in 2002 and 2005, respectively. Currently, he is an associate professor in the School of Information Science and Technology, University of Science and Technology of China. His research interests include information hiding and multimedia security.