INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume 4, Issue 2)
Available online at: www.ijariit.com

A Novel Scheme for Separable and Reversible Data Hiding in Encrypted JPEG Image using Forward Correction Code

Bushra MP
mp.bushra@gmail.com
Cochin College of Engineering and Technology
Valanchery, Malappuram, Kerala

Naseeda P
Nasheedakhalid123@gmail.com
Cochin College of Engineering and Technology
Valanchery, Malappuram, Kerala

ABSTRACT

Among various digital image formats used in daily life, JPEG is the most popular. Therefore, reversible data hiding (RDH) in JPEG images is important and useful for many application such as archive management and image authentication. Reversible data hiding in encrypted images (RDH-EI) are developed most techniques for uncompressed images, this paper provides a separable reversible data hiding protocol for encrypted JPEG bitstreams. Here a JPEG encryption algorithm is first proposed, which enciphers an image to a smaller size and keeps the format compliant with 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 JPEG bitstream without changing the bitstream size. On the recipient side, the original bitstream can be reconstructed losslessly using an iterative recovery algorithm based on the blockingartifact. Though message extraction and image recovery are separable, anyone who has the embedding key can extract the message from the marked encrypted copy. A random error is introduced to obtain the PSNR and error values of different JPEG images and make a comparative study. A forward error correction technique is used for controlling errors. Experimental results show that the proposed method outperforms a previous work in terms of separation capability, embedding capacity and security.

Keywords: *Jpeg Encryption, Data Hiding Information, Jpeg Decryption, Security.*

1. INTRODUCTION

Signal processing in the encrypted domain (SPED) for privacy preserving has attracted considerable research interests in recent years. 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. Many approaches have been proposed for different applications, for example, compressing encrypted images, signal transformation in cipher texts, pattern recognition in encrypted domain, watermarking in encrypted multimedia, data searching in encrypted dataset etc. Reversible data hiding in encrypted images (RDH-EI) is another topic of SPED.

Motivations and Contributions

RHD-EI allows a server to embed an 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. 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.

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. This paper first proposes 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.

2. LITERATURE REVIEW

A literature survey is done for specified papers which are essential to know the existing techniques their significance and limitations. It also includes various supporting papers for the proposed technique and their advantages. Here all the papers are viewed based on its algorithm architectural flow.

3. BACKGROUND WORK

Reversible Data Hiding in Encrypted Image

Reversible data hiding is a technique to embed an additional message into some distortion-unacceptable cover media, such as military or medical images, with a reversible manner so that the original cover content can be perfectly restored after extraction of the hidden message. A number of reversible data hiding methods have been proposed in recent years. This work proposes a novel reversible data hiding scheme for the encrypted image, which is made up of image encryption, data embedding, and data-extraction/image-recovery phases. The data of original cover are entirely encrypted, and the additional message is embedded by modifying a part of encrypted data. At the receiver side, with the aid of spatial correlation in a natural image, the embedded data are successfully extracted while the original image is perfectly recovered.

A. Efficient Compression of Encrypted Gray Scale Images

This paper shows an efficient way to compress encrypted images through resolution-progressive compression (RPC). The encoder starts by sending a down sampled version of the cipher text. At the decoder, the corresponding low-resolution image is decoded and decrypted, from which a higher-resolution image is obtained by intra frame prediction. The predicted image, together with the secret encryption key, is used as the side information (SI) to decode the next resolution level. This process is iterated until the whole image is decoded. By doing so, the task of de-correlating the pixels, which is not possible for the encoder, is shifted to the decoder side. In addition, by having access to a lower-resolution image, the decoder is able to learn the local statistics, doing much better than “blind” decoding. Moreover, by avoiding exploiting the Markovian property in Slepian-Wolf decoding, the decoder’s complexity is significantly reduced.

B. Reversible Data Hiding in Encrypted JPEG Bitstream

Reversible data hiding (RDH) is a technique that embeds secret information into a cover image in a reversible manner. On the receiving side, the hidden message can be extracted and the original image perfectly restored. This technique is especially useful in applications such as medical and military imaging where the original image must not be altered after the embedded data are extracted. Unlike robust watermarking, RDH emphasizes perfect image reconstruction and data extraction, but not the robustness against malicious attacks. This is useful, for example, in cloud storage. When a database manager tries to label the files using RDH methods, no transmission is involved, therefore no errors and attacks either. This correspondence proposes a framework of reversible data hiding (RDH) in an encrypted JPEG bitstream. Unlike existing RDH methods for encrypted spatial-domain images, the proposed method aims at encrypting a JPEG bitstream into a properly organized structure and embedding a secret message into the encrypted bitstream by slightly modifying the JPEG stream. We identify usable bits suitable for data hiding so that the encrypted bitstream carrying secret data can be correctly decoded. The secret message bits are encoded with error correction codes to achieve a perfect data extraction and image recovery. The encryption and embedding are controlled by encryption and embedding keys respectively. If a receiver has both keys, the secret bits can be extracted by analyzing the blocking artifacts of the neighboring blocks, and the original bitstream perfectly recovered. In case the receiver only has the encryption key, he/she can still decode the bitstream to obtain the image with good quality without extracting the hidden data.

C. Reversible Data Hiding in Encrypted Images by Reserving Room Before Encryption

Reversible data hiding (RDH) in images is a technique, by which the original cover can be losslessly recovered after the embedded message is extracted. This important technique is widely used in medical imagery, military imagery, and law forensics, where no distortion of the original cover is allowed. Since first introduced, RDH has attracted considerable research interest. Recently, more and more attention is paid to reversible data hiding (RDH) in encrypted images, since it maintains the excellent property that the original cover can be losslessly recovered after embedded data is extracted while protecting the image content’s confidentiality. All previous methods embed data by reversibly vacating room from the encrypted images, which may be subject to some errors in data extraction and/or image restoration. In this paper, we propose a novel method by reserving a room before encryption with a traditional RDH algorithm, and thus it is easy for the data hider to reversibly embed data in the encrypted image. The proposed method can achieve real reversibility, that is, data extraction and image recovery are free of any error. Experiments show that this novel method can embed more than 10 times as large payloads for the same image quality as the previous methods, such as for PSNR= 40 dB.

D. Protection and Retrieval of Encrypted Multimedia Content: When Cryptography Meets Signal Processing

In the past few years, the processing of encrypted signals has emerged as a new and challenging research field. The combination of cryptographic techniques and signal processing is not new. So far, encryption was always considered as an add-on after signal manipulations had taken place. For instance, when encrypting compressed multimedia signals such as audio, images, and video, first the multimedia signals were compressed using state-of-the-art compression techniques, and next encryption of the compressed bit stream using a symmetric cryptosystem took place. Consequently, the bit stream must be decrypted before the multimedia signal can be decompressed. An example of this approach is JPSEC, the extension of the JPEG2000 image compression standard. This standard adds selective encryption to JPEG2000 bit streams in order to provide secure scalable streaming and secure transcoding.

Conclusion of Literature Review

All the above papers described reversible data hiding system that consists of image encryption, data embedding, and data extraction/ image-recovery phases. The data of the original image are entirely encrypted by a stream cipher. Although a data-hider does not know the original content, he can embed additional data into the encrypted image by modifying a part of encrypted data. With an encrypted image containing embedded data, a receiver may firstly decrypt it using the encryption key, and the decrypted version is similar to the original image. According to the data-hiding key, with the aid of spatial correlation in a natural image, the embedded data can be correctly extracted while the original image can be perfectly recovered. Although someone with the knowledge of encryption key can obtain a decrypted image and detect the presence of hidden data using LSB-steganalytic methods.

4. PROPOSED WORK

The framework of the proposed method is depicted in Fig. 1. 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.

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.

A. JPEG Encryption

Before encryption, the content owner parses the JPEG bitstream according to the simplified syntax of baseline, as shown in Fig. 2. 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. 2 indicates that the entropy encoded data contains entropy-coded segments of all blocks. If a grayscale image sized H×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 ECSi i=1, 2, …, N.

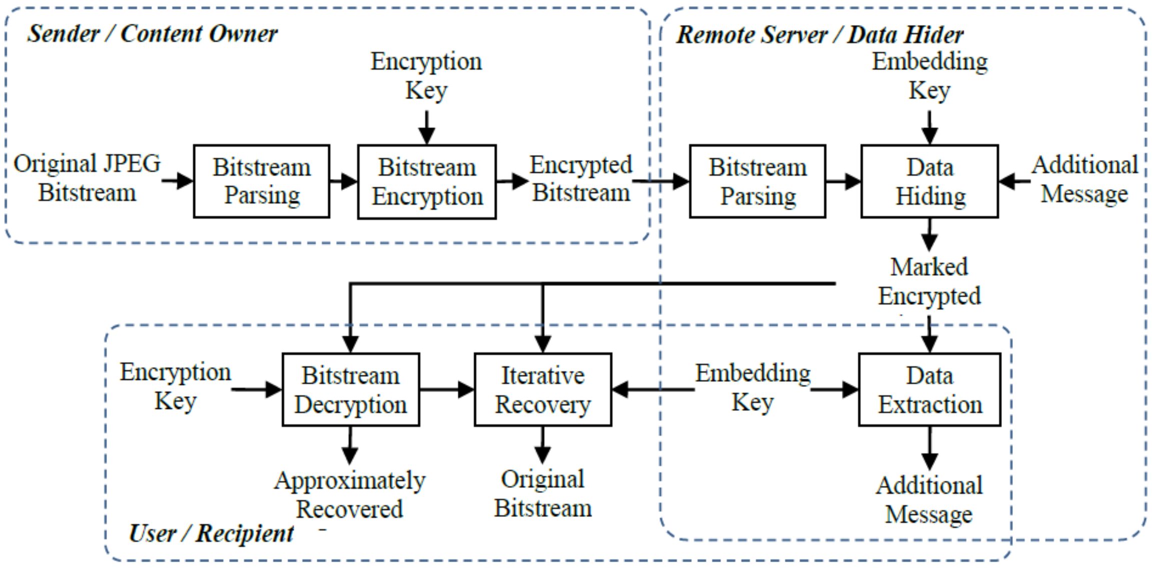


Fig. 1 Framework of the Proposed Method

Each entropy-coded segment contains codes of DC and AC coefficients, as shown in the third layer of Fig. 2. Denote the codes of DC and AC coefficients in the *i*-th segment as **DCC**<*i*> and **ACC**<*i*, *j*>, respectively, where *i*=1, 2, …, *N* and 0 ≤ *j* < 64. In the fourth layer, both codes of DC and AC coefficients contain Huffman code and appended bits. Let **DCH**<*i*> and **DCA**<*i*> be the Huffman codes and appended bits for DC coefficient, **ACH**<*i*, *j*> and **ACA**<*i*, *j*> for AC coefficient, respectively. Thus, each entropy-coded segment can be represented by

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

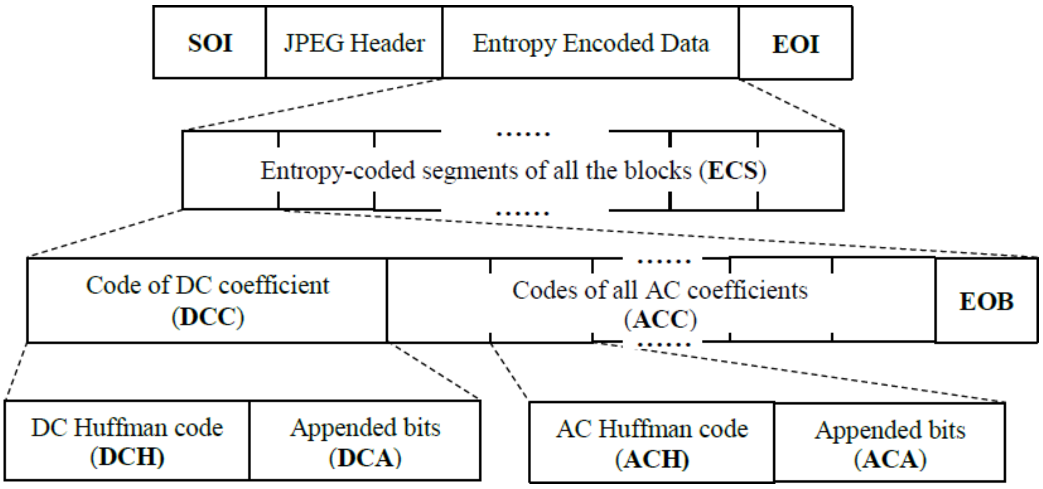


Fig. 2 Simplified Syntax of JPEG Baseline. SOI, EOI, and EOB Stand for Start-of-Image, End-of-Image and End-of-Block Respectively. Acronyms in the Parentheses are used in the Discussion for Brevity

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-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. 3. 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

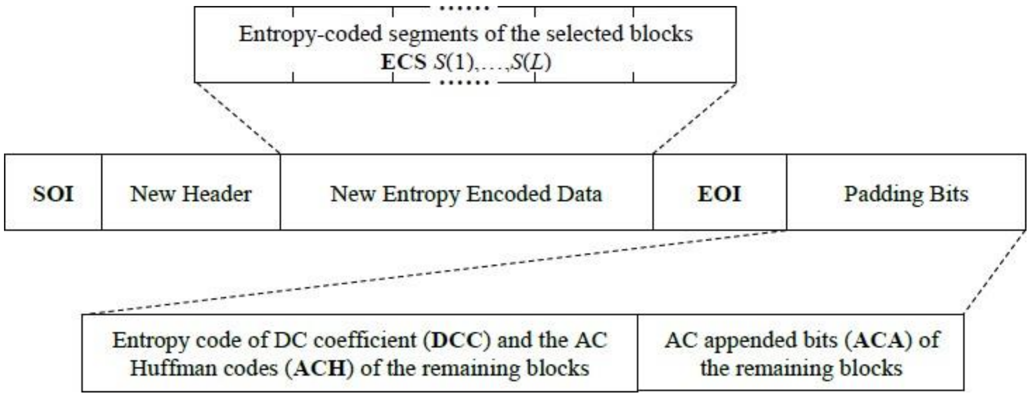

Entropy-coded segments of the selected blocks
ECS S(1),…,S(L)
SOI
New Header
New Entropy Encoded Data
EOI
Padding Bits
Entropy code of DC coefficient (DCC) and the AC Huffman codes (ACH) of the remaining blocks
AC appended bits (ACA) of the remaining blocks

Fig. 3 The syntax of the New JPEG Bitstream

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

where,

$$\mathbf{CC} = \{\{\mathbf{DCC}<R(1)>, \mathbf{ACH}<R(1), 1>, \mathbf{ACH}<R(1), 2>, \ldots, \mathbf{EOB}\}, \ldots, \{\mathbf{DCC}<R(N–L)>, \mathbf{ACH}<R(N–L), 1>, \mathbf{ACH}<R(N–L), 2>, \ldots, \mathbf{EOB}\}\}$$

$$\mathbf{AP} = \{\{\mathbf{ACA}<R(1), 1>, \mathbf{ACA}<R(1), 2>, \ldots\}, \ldots, \{\mathbf{ACA}<R(N–L), 1>, \mathbf{ACA}<R(N–L), 2>, \ldots\}\}$$

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 of the padding data and **P**=[*p*1, *p*2, …, *pM*]. With the encryption key *K*enc again, the content owner generates a key stream **K**=[*k*1, *k*2, …, *kM*] using a stream cipher algorithm such as RC4 and SEAL. The padding bits are then encrypted to **P**'=[p_1', p_2', …, p_M'] where

$$p_i' \square p_i \square k_i, 1 \square i \square M$$

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 P' and the parameters H and W into the reserved application segments, marked as APPn in the JPEG header, in the same way. After the processing, an encrypted JPEG bitstream is generated. The encrypted bitstream has the same amount of data as the original and is compliant with 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×w using commonly-used JPEG decoders.

An example of the proposed JPEG encryption is shown in Fig. 3.4, 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. 3.4(b), 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.

B. 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 data can be extracted by parsing the bitstream. Meanwhile, the padding bits **P**'=[p_1', p_2',…, p_M'] can be extracted from the reserved application segments marked as **APP$_n$** in the JPEG header. With the encryption key Kenc, the appended bits of L selected entropy-coded segments and the padding bits P={CC, AP} can be deciphered in the same way as (1).

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 ECS$_u$, u=R(1), R(2), …, R(N–L). Meanwhile, the L selected entropy-coded segments ECS$_v$ (v=S(1), S(2), …, 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 **ECS**$_u$ (u=R(1), R(2), …, R(N–L)) and the remaining **ECS**$_v$ (v=S(1), S(2), …, S(L)) are sequentially put back to the original positions, and the JPEG header is modified to restore the original image size H×W. The decryption procedure is depicted in Fig. 5.

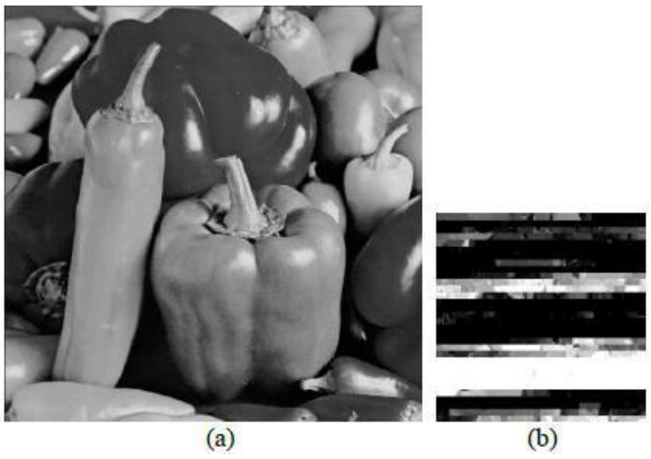


Fig. 4 Encryption of JPEG Bitstream: (a) Image Decoded from the Plaintext JPEG Bitstream; (b) Image Decoded from the Encrypted Bitstream

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. 6. 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 subsection.

The data hider evenly divides the binary vector **A** into s groups {$\mathbf{A}_1$, $\mathbf{A}_2$, …, $\mathbf{A}_s$}, where s=m/n, n=β·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 is ma such bits, the value of e is calculated by e=m_a/L.

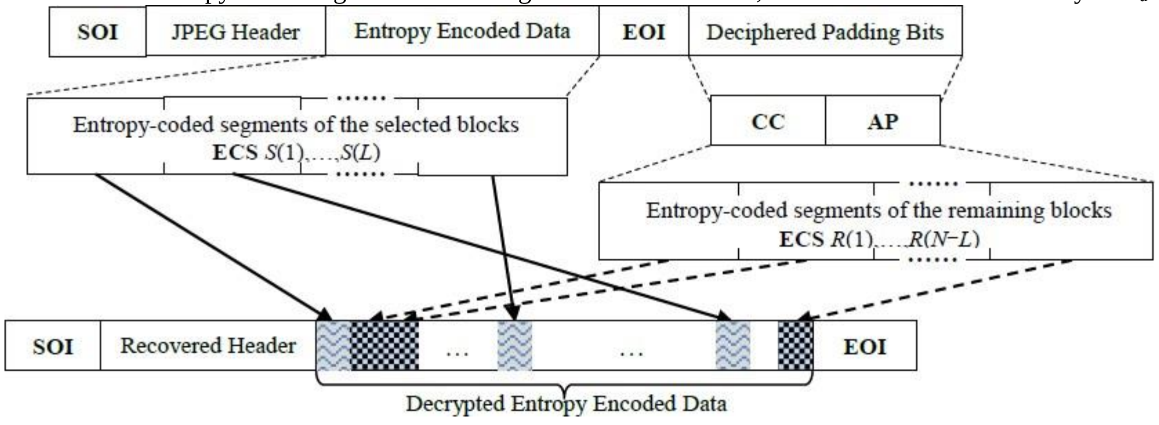


Fig. 5 Procedure of JPEG Decryption

Subsequently, the data hider constructs a k×n binary matrix H by

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

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 **A***t* (*t*=1, 2, …, *s*), the data hider further computes

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

in which all calculations are binary arithmetic. Thus, each group **A***t* containing *n* bits is compressed to a vector **B***i* with *k* bits. After compressing all groups from *m* bits to *ks* bits, the additional message **M** containing *m*–*ks* bits are appended to generate\

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

With an embedding key *K*emb, the data hider shuffles **P***m* to produce a sequence **E**. These bits are then embedded into the reserved application segments marked as **APPₙ** in JPEG header. This way, a *marked encrypted JPEG bitstream* containing additional messages is generated.

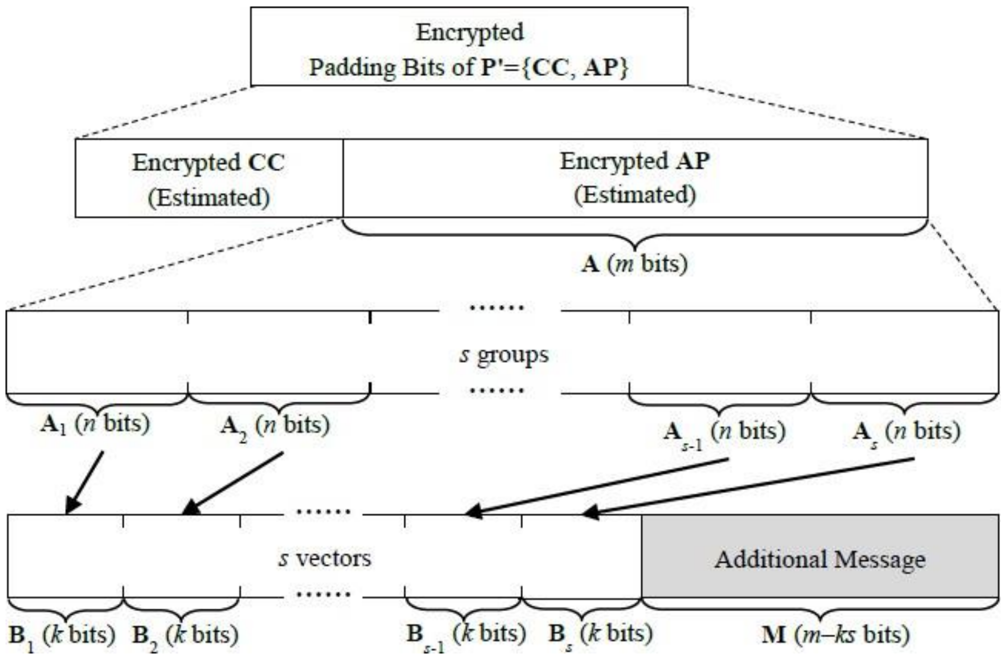


Fig. 6 Procedure of Data Embedding

To extract the additional messages, one can reshuffle the padding bits of the marked encrypted bitstream using the embedding key Kemb, 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 APPn in JPEG header. Another is to estimate the length of encrypted padding bits. If there are ns bits in L selected entropy-coded segments and nr bits in the N–L remaining segments, m can be estimated by m=[λ·(nr/ns)·ma], where [·] is a rounding operator, and λ a scaling factor (0<λ<1) used to avoid modifying the encrypted Huffman codes during data hiding.

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 III-A. 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. 7. The recipient first parses and extracts the encrypted padding bits **E** from the marked encrypted JPEG bitstream and reshuffles **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 H, the recipient generates a binary matrix G,

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

where *r*=*n*–*k* and *n*=*βe*. For each group **B***t*, the recipient calculates

$$[\mathbf{A}_t^{(c)}(\mathbf{1}), \mathbf{A}_t^{(c)}(\mathbf{2}) \ldots \mathbf{A}_t^{(c)}(\boldsymbol{n})]$$

$$= [\mathbf{B}_t\ (1), \mathbf{B}_t\ (2) \ldots \mathbf{B}_t\ (k), \mathbf{0} \ldots \mathbf{0}] + [a_1, a_2, \ldots, a_r] \cdot \mathbf{G}$$

where,

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

With the encryption key *K*enc, the candidate vectors $\mathrm{A}_t^{(c)}$ are decrypted to plaintext candidates $\mathrm{D}_t^{(c)}$ 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 $\mathrm{D}_t^{(c)}$ belongs to several entropy-encoded segments, *i.e.*,

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

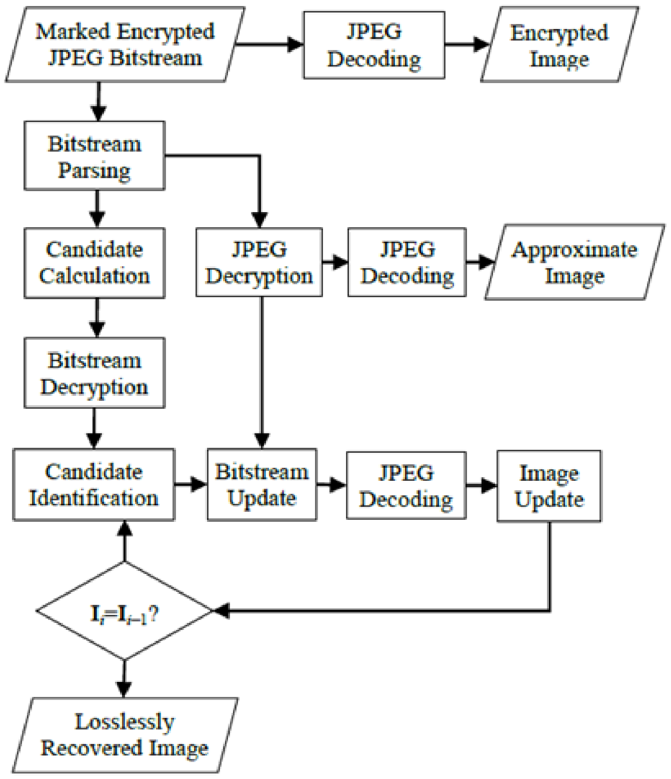


Fig. 7 Flow Chart of Iterative Recovery

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

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

where $\mathbf{ACA}^{<ta,\, j>}$ is a candidate of appended bits, and 0≤j<64. Thus, 2^r candidates for l entropy-coded segments can be generated,

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

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

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

$$\mathbf{b}_t = \arg_c \min \sum_{a=t1}^{tl} f(\mathbf{PB}_a^{(c)})$$

and f is the blocking artifact function,

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

The blocking artifact function is illustrated in Fig. 8, 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. 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 **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 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,…,2^r), the recipient iteratively finds the best candidates.

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

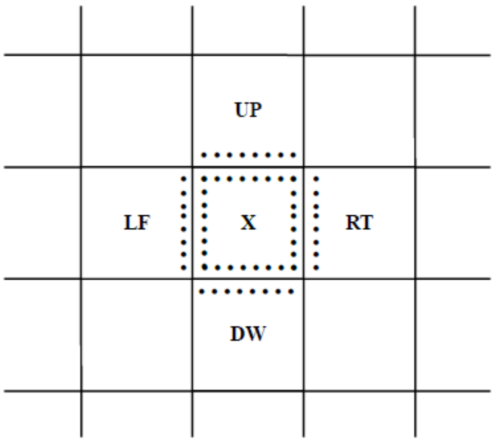


Fig. 8. Blocking Artifacts

As shown in Fig. 8, **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.

5. EXPERIMENTAL ANALYSIS

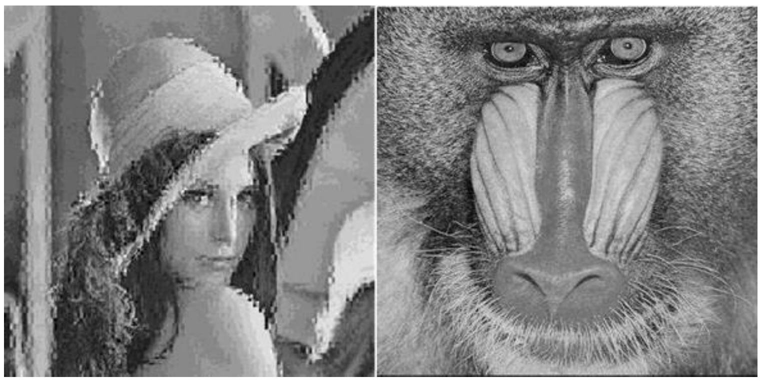
To verify the proposed method, we use a grayscale image sized 512×512, and compress them to JPEG bitstreams with different quality factors. The encrypted padding bits and the parameters m, H, and W are hidden into the JPEG header. Let α=L/N be the ratio of selected blocks.

An example is given in Fig. 9, in which shows the original images compressed with a quality factor 80. The encrypted bitstreams have the same lengths as the original. Secret messages containing 1023 bits are embedded into each encrypted bitstream using an embedding key Kemb. On the recipient side, additional messages can be extracted without any errors if the error value is zero otherwise there is an error. The key Kemb is available. When both keys are available, the original bitstreams can be recovered without loss.

The quality factor used for scaling the default quantization table is an important factor in JPEG compression, and it also impacts embedding payloads.

(a)

(b)


(c)

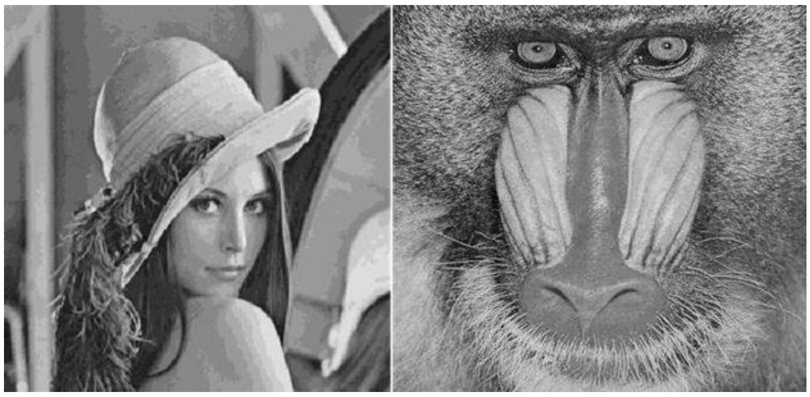

(d)

Fig. 9. RDH-EI in the Bitstreams Corresponding to Lena and Baboon: (a) Original JPEG Images, (b) Encrypted Images, (c) Directly Decrypted Images, and (d) Losslessly Recovered Images

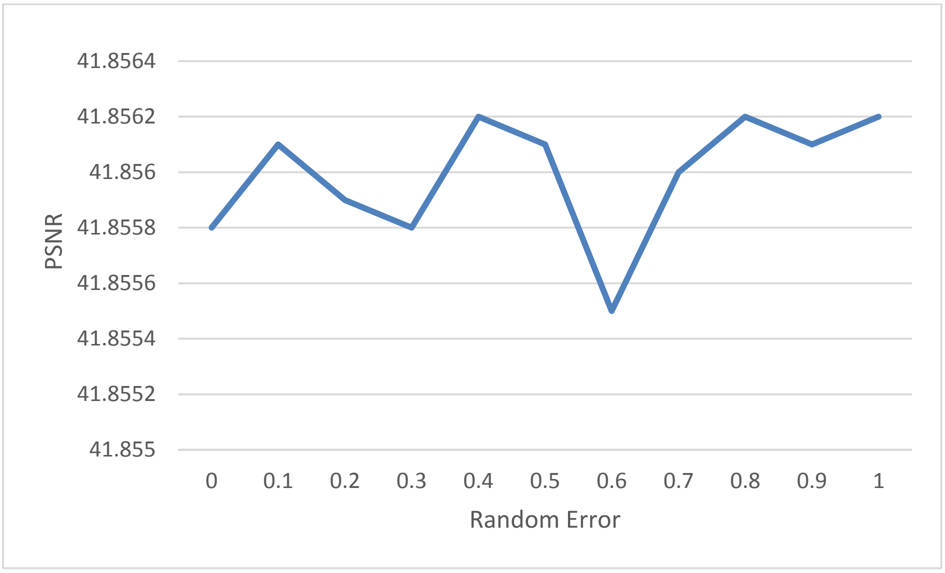
If an adversary uses all Huffman codes and sets all appended bits to zero, a contour of the original image can be revealed. While in the proposed method, only part of the bitstream segments is randomly selected to construct a smaller sized JPEG image. most of the entropy-coded segments are rearranged 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.A forward error correction technique is used to control the error.

6. RESULT AND DISCUSSION

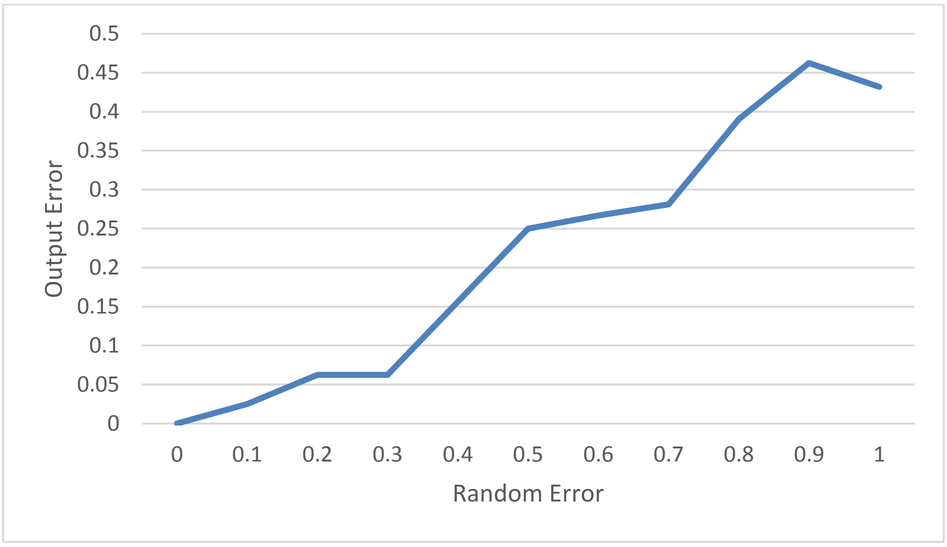
The following figures depict the simulation results. Figure 9 shows the losslessly recovered images of Lena and Baboon.The PSNR and output error is obtained using forward error correction method.

Table 1. Values of PSNR and Output Error for Different Random Errors for Lena

Random Error	PSNR	Output Error
0	41.8558	0
0.1	41.8561	0.025
0.2	41.8559	0.0625
0.3	41.8558	0.0625
0.4	41.8562	0.1563
0.5	41.8561	0.25
0.6	41.8555	0.2667
0.7	41.856	0.2813
0.8	41.8562	0.3906
0.9	41.8561	0.4625
1	41.8562	0.4318



(a)

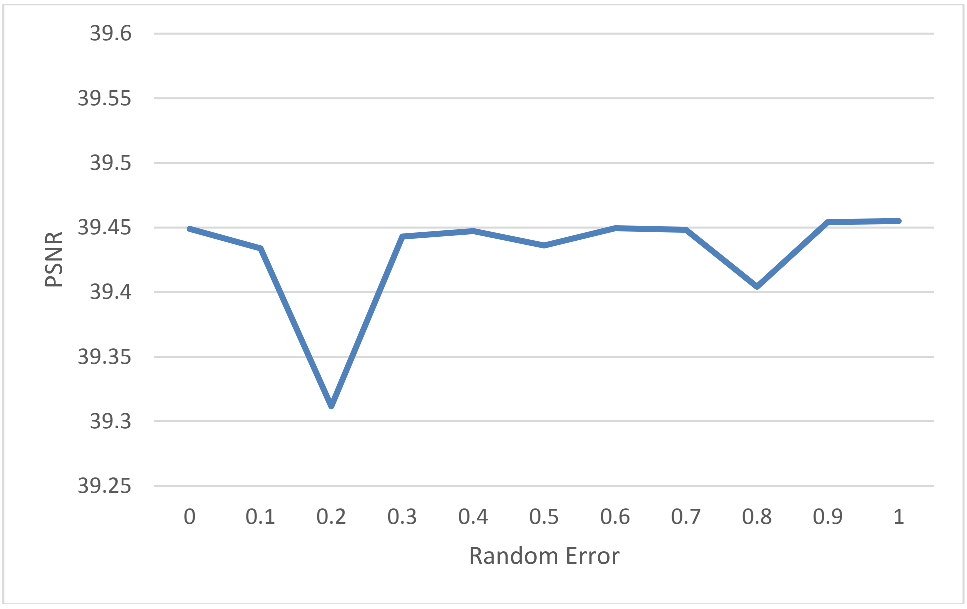


(b)

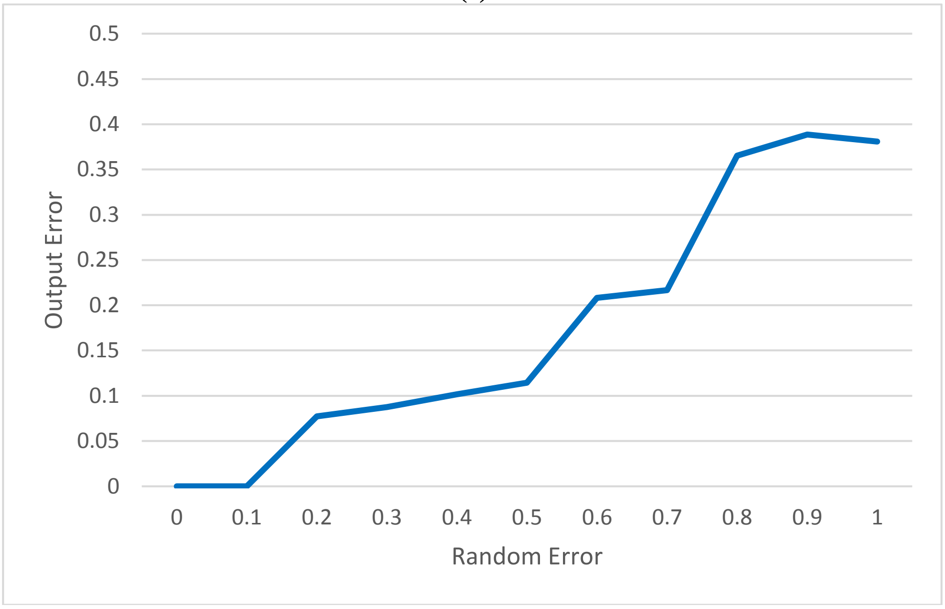
Fig. 10. (a) Graphical Representation of Output Error Verses Random Error (b) Graphical Representation of PSNR Verses Random Error of the Image Lena

Table 2.Values of PSNR and output Error for Different Random Errors for Baboon

Random Error	PSNR	Output Error
0	39.449	0
0.1	39.4338	0
0.2	39.3116	0.0774
0.3	39.443	0.0875
0.4	39.4473	0.1017
0.5	39.436	0.1143
0.6	39.4495	0.2083
0.7	39.4483	0.2167
0.8	39.404	0.3654
0.9	39.4541	0.3886
1	39.455	0.3808

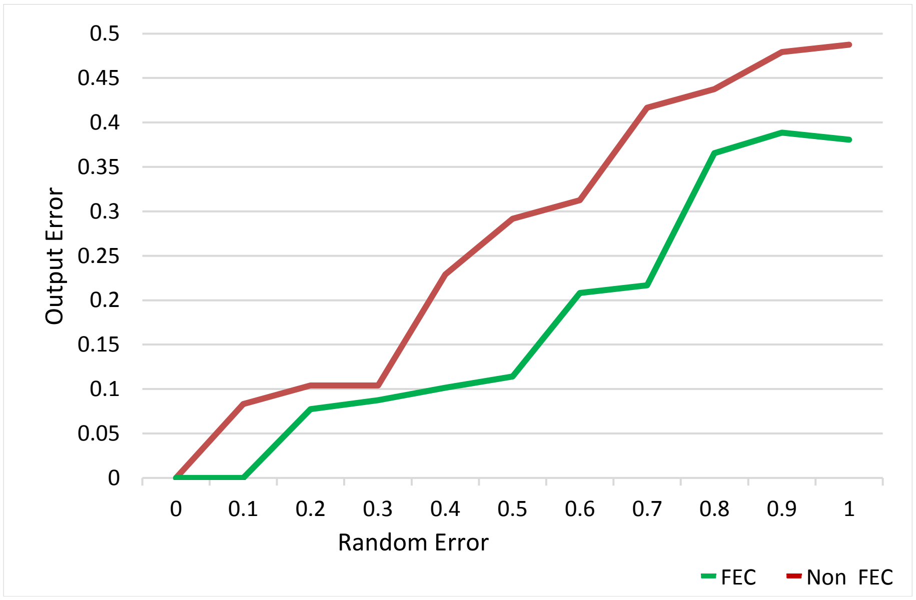


(a)

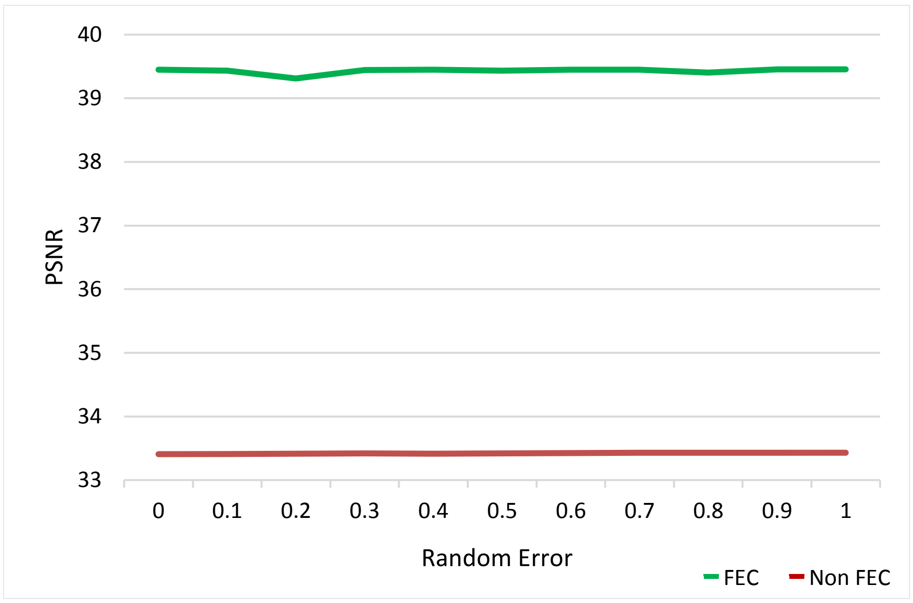


(b)

Fig 11(a) Graphical Representation of PSNR Verses Random Error (b) Graphical Representation of Output Error Verses Random Error of the Image Baboon

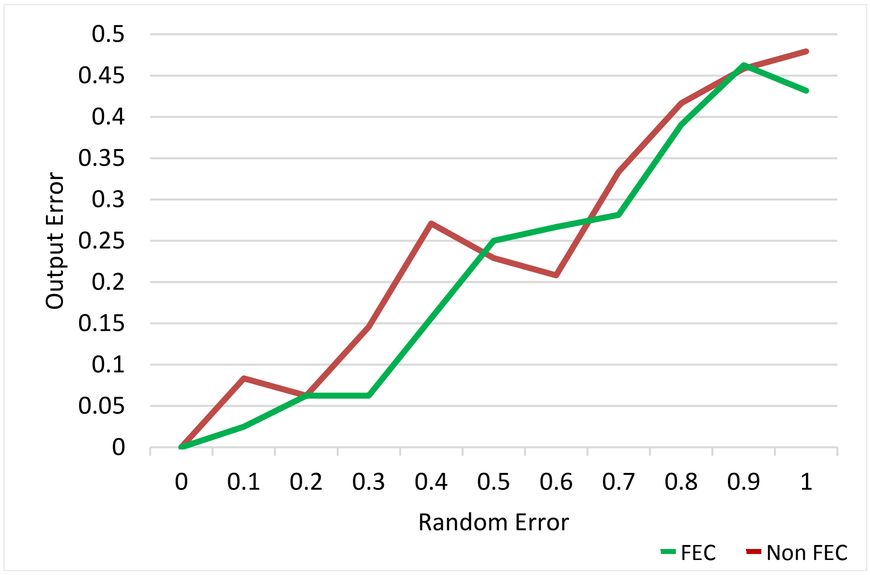


(a)

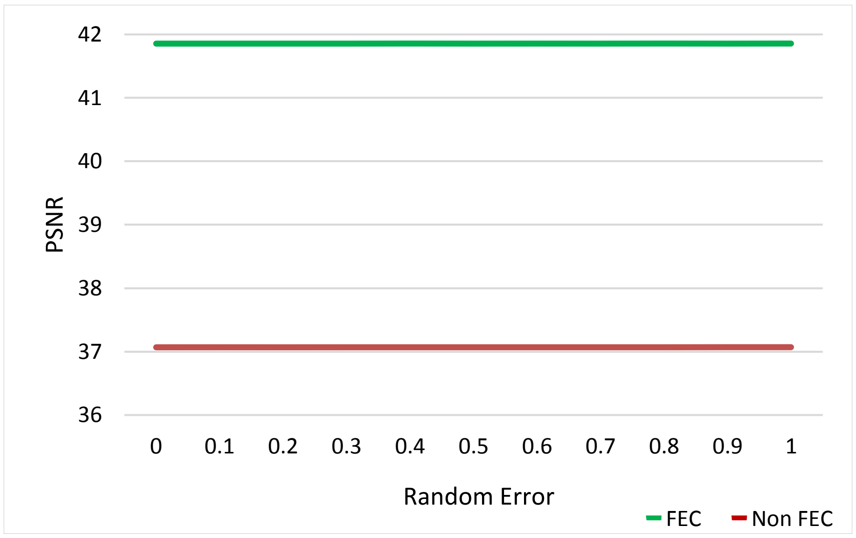


(b)

Fig 12 (a) Output Error Comparison for Baboon (b) PSNR Comparison for Baboon



(a)



(b)

Fig 13 (a) Output Error Comparison for Lena (b) PSNR Comparison for Lena

7. CONCLUSION

This paper proposes a separable reversible data hiding scheme for the encrypted JPEG bitstream. A JPEG encryption and decryption algorithm are 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., the 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 $\boldsymbol{APP_n}$ 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 bit stream is unchanged. This paper concludes that using forward error correction method it provides better PSNR and controls the bit error.

8. REFERENCES

[1] Z. Qin, X. Zhang, and S. Wang, "Reversible data hiding in encrypted JPEG bitstream," *IEEE Trans. Multimed.*, vol. 16, no. 5, pp. 1486-1491, 2014.
[2] X. Zhang, Z. Qian, G. Feng, and Y. Ren, "Efficient reversible data hiding in encrypted images," *J. Visual Comm. Image Representation*, vol. 25, no. 2, pp. 322-328, 2014..
[3] W. Zhang, K. Ma, and N. Yu, "Reversibility improved data hiding in encrypted images," *Signal Process,* vol. 94, pp. 118-127, 2014
[4] K. Ma, W. Zhang, X. Zhao, N. Yu, and F. Li, "Reversible data hiding in encrypted images by reserving a room before encryption," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 3, pp. 553-562, 2013
[5] 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, 2012.
[6] X. Zhang, "Separable reversible data hiding in encrypted image," *IEEE Trans. Inf. Forensics Security,* vol. 7, no. 2, pp. 826-832, 2012.
[7] X. Zhang, "Reversible data hiding in encrypted image," *IEEE Signal Process. Lett.*, vol. 18, no. 4, pp. 255-258, 2011.
[8] W. Liu, W. Zeng, L. Dong, and Q. Yao, ―Efficient compression of encrypted grayscale images,‖ *IEEE Transactions on Image Processing,* vol. 19, no. 4, pp. 1097-1102, 2010
[9] Z. Ni, Y. Q. Shi, N. Ansari and W. Su, "Reversible data hiding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 3, pp. 354-362, 2006.
[10] J. Fridrich, M. Goljan, and R. Du, "Invertible authentication," *Proc. SPIE,* San Jose, CA, vol. 3971, pp. 197-208, 2001.
[11] J. Fridrich, M. Goljan, and R. Du, "Lossless data embedding: a new paradigm in digital watermarking," *EURASIP J. Appl. Signal Process.*, vol. 2, pp. 185-196, 2002.
[12] M. U. Celik, G. Sharma and A. M. Tekalp, "Lossless watermarking for image authentication: a new framework and an implementation," *IEEE Trans. Image Process.*, vol. 15, no. 4, pp. 1042-1049, 2006.
[13] J. Tian, "Reversible data embedding using a difference expansion," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 8, pp. 890-896, 2003.
[14] D. M. Thodi and J. J. Rodríguez, "Expansion embedding techniques for reversible watermarking," *IEEE Trans. Image Process.*, vol. 16, no. 3, pp. 721-730, 2007.
[15] P. Tsai, Y. Hu and H. Yeh, "Reversible image hiding scheme using predictive coding and histogram shifting," *Signal Process.*, vol. 89, no. 6, pp. 1129-1143, 2009.
[16] Z. Erkin, A. Piva, S. Katzenbeisser, et al., ―Protection and retrieval of encrypted multimedia content: when cryptography meets signal processing,‖ *EURASIP Journal on Information Security*, 2008.
[17] V. Sachnev, H. Joong Kim, J. Nam, S. Suresh, and Y. Q. Shi, "Reversible watermarking algorithm using sorting and prediction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 7, 2009. 670, Oct. 2002.