https://doi.org/10.1016/j.jvcir.2018.11.018

Joint Image Encryption and Compression Schemes Based on 16×16 DCT

PEIYA LI, The Hong Kong Polytechnic University
KWOK-TUNG LO, The Hong Kong Polytechnic University

Joint image encryption and compression schemes have shown their great potential values in protecting compressed images, which are the most common form of Internet images. To achieve the protection, a trade-off between encryption ability and compression ability needs to be considered. To satisfy different encryption/compression requirements, in this paper, we propose two new joint encryption and compression schemes by incorporating scrambling techniques in different intermediate stages of the JPEG process, where one scheme emphasizes compression performance and encryption efficiency, another highlights protection performance. In order to enhance the protection ability against the chosen-plain-text attack, in both schemes, we use adaptive key for each plainimage's encryption/decryption, and the key is dependent on the plainimage. The initial encryption operation we make in the two proposed schemes is to replace JPEG's original 8×8 DCT with 16×16 DCT, which means that we raster scan the plain-image into non-overlapping 16×16 blocks, and the following encryption operations are all based on it. In the first encryption scheme, after applying order-16 DCT for all 16×16 blocks, we do block permutation, and DC coefficients confusion for encryption, all controlled by the adaptive key. As for our second encryption scheme, to further improve the protection power, we add the run/size and value (RSV) pairs' shuffling operation in the entropy coding stage, on the basis of the first scheme. Performance evaluations on these two encryption schemes using various criteria are conducted, and the results have shown that the first scheme has better compression performance and higher computational efficiency, while the second scheme has better defense ability against the differential attack and statistical attack.

Additional Key Words and Phrases: Image encryption, JPEG compression, 16×16 DCT, differential attack, statistical attack

ACM Reference format:
Peiya Li and Kwok-Tung Lo. 2017. Joint Image Encryption and Compression Schemes Based on 16×16 DCT. 1, 1, Article 1 (April 2017), 18 pages.
DOI: 0000001.0000001

1 INTRODUCTION

Due to the drastic development of network technology, approaches for communication have been taken to new era. People communicate with each other anywhere, any-time through various devices, such as smart mobile phones, laptops, and personal computers. Using these devices, multimedia content can be easily generated and transmitted over the Internet to the specific people/group. However, this easy access mode and distribution convenience also increase the risk of eavesdropping and intercepting when sensitive multimedia data are sent and received. Hence, securing multimedia data through encryption has become more common to be adopted to guarantee the safety of these multimedia contents.

As an important research area of multimedia data protection, image encryption has four major objectives that will help evaluating and comparing different image encryption algorithms. These four objectives are listed as follows:

Encryption security: This includes both perceptual security and cryptographic security (Lian 2008). The former refers to the perceptual distortion of cipherimage with respect to the plainimage, which is commonly measured by the peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) (Wang and Bovik 2002). The latter represents the encryption scheme's resistance ability against cryptographic attacks, such as brute-force attack, differential attack, statistical attack, etc.

Format compliance: Format information is generated after image data is encoded using some compression algorithms. This information will be used by decoder to successfully recover the compressed data and to keep the communication synchronized between the encoder and decoder (Lian 2008). When applying encryption algorithms to images, the encrypted bit-stream should be compliant with the compressor, which means that the encrypted bit-stream should be decoded by any standard decoder even without decryption. This property is essential since it can preserve some good features of the corresponding compression standard (Massoudi et al. 2008).

Compression friendliness: Most of the images we see on Internet are compressed, thus in all cases, encryption algorithms should have no or very little influence on the compression efficiency. Some algorithms may introduce some overheads which is necessary for decryption, but the impact of these overheads on the final compression ratio should be limited (Massoudi et al. 2008; Xu et al. 2014).

Encryption efficiency: Unlike text data, the size of image data is often very huge, encrypting these data in an efficient way is particularly important for the real-time processing requirement (Xu et al. 2014).

Joint Photographic Experts Group (JPEG) is a commonly used method of lossy compression for digital images, it can achieve a good trade-off between image quality and storage size. The general architecture for JPEG compression standard is shown in Figure 1. From Figure 1, we can see that there are mainly four stages of JPEG, transformation stage, quantization stage, zig-zag scan stage, and entropy coding stage. The lossy compression is realized at the quantization stage, which is controlled by the quality factor (QF) to adjust compression ratio. Though JPEG can greatly save the storage space and transmission bandwidth, it does not offer any protection ability for the images (Zhang and Zhang 2014). Therefore, on the basis of studying and analysing the compression procedure of JPEG image, many encryption algorithms were proposed to provide protection for this kind of images. Considering the different potential positions for the encryption algorithms to be embedded into the compression process, JPEG image encryption can be divided into three categories: pre-compression encryption, in-compression encryption, and post-compression encryption.

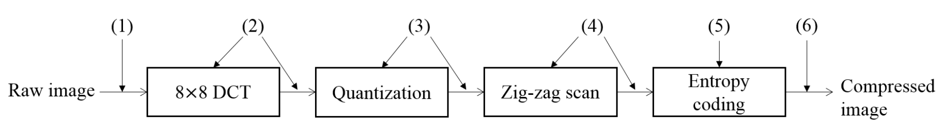


Fig. 1. Block diagram of JPEG baseline coder.

Pre-compression encryption means to perform encryption before compression, which is realized at position (1) of Figure 1. Encryption algorithms, like the permutation-only encryption methods (Indrakanti and Avadhani 2011; Mitra et al. 2006; Younes and Jantan 2008) or some chaos-based encryption schemes (Faragallah 2015; Kumar et al. 2015; Ponnain and Chandranbabu 2016) which are directly operated on raw images, can be classified into this category. However, this class of algorithms is generally inapplicable for lossy compression, because under lossy compression, pixel value in cipherimage cannot be fully recovered to the same value as the plainimage. Additionally, performing encryption before compression often destroys the correlation in plainimage, little space can be exploited for compression, thus pre-compression encryption schemes are usually not compression friendly.

In-compression is to perform encryption and compression jointly, which can be implemented at positions (2)-(5) of Figure 1. Encryptions realized at various intermediate stages are listed as below:

Encryption at position (2): In (Yeung et al. 2009, 2011a,b; Zeng et al. 2014), they realized partial video encryption at the transformation stage by replacing the original 4×4 or 8×8 DCT with new orthogonal transforms, which could also be applied for JPEG image encryption. This encryption scheme could achieve good protection performance without sacrificing compression efficiency too much, but its protection degree was limited because they only modified the transformation stage for encryption, and the processing unit remained the 4×4 or 8×8 block, little diffusion property was possessed by this encryption idea, which was vulnerable to the differential attack.

Encryption at position (3): For encryption at quantization stage, (Ong et al. 2015; Qian et al. 2014) proposed to change the magnitude of entries in quantization table (QT), however only the quantization table change may not offer enough encryption security, it often accompanies with other stages' encryption, and the changed entries' magnitudes may have an impact on the final compression ratio.

Encryption at position (4): In (Jha 2014; Maniccam and Bourbakis 2004), they proposed a joint encryption and compression system based on the SCAN language, which is a family of formal languages-based two-dimensional spatial accessing methodology, was capable of representing and creating a great variety of 2-D array scanning paths from a small set of primitive ones. This type of encryption scheme has high security level with good confusion and diffusion properties, but its main drawback is that the high computational overhead will cause delay in the compression/encryption procedure and is not so compression friendliness.

Encryption at position (5): Encryption algorithms realized at the entropy-coding stage include using multiple Huffman tables (Wu and Kuo 2005), shuffling the identifiers of zig-zag scan encoded sequences (Ji et al. 2015), etc. Nonetheless, these algorithms suffered from the not format compliant problem.

Encryption algorithms from post-compression class perform encryption after compression, which corresponds to position (6) in Figure 1. Xu *et al.*(Xu et al. 2014) proposed to encrypt the JPEG compressed data stream using the variable modular encryption method, which was based on spacing mapping. This scheme was compression friendly, because it mapped each codeword to another codeword with same code length. But it may confront the format non-compliance problem, since adopting its mapping strategy, the number of elements in each 8×8 block may exceed 64.

In our work, we propose two different encryption methods to achieve joint encryption and compression for JPEG image. Both of them are format-compliant to JPEG standard, and are performed in the intermediate stages of JPEG, which mainly focus on positions (2), (3) and (5) of Figure 1. To enhance the diffusion ability of the two cryptosystems with respect to the plainimage, we use adaptive keys for different images' encryption, and the key is plainimage-dependent. Our first scheme mainly includes three encryption techniques: 16×16 blocks' transformation using order-16 DCT, 8×8 blocks' permutation, and DC coefficients confusion. It can maintain the good compression ability of JPEG, and meanwhile offer a certain level of protection. For the second scheme, we add the Run/Size and Value (RSV) pairs shuffling operation in the entropy coding stage, thus it has higher protection level, but the compression performance and encryption efficiency are little compromised. Therefore, the proposed two schemes are suitable for different applications. For applications paying more attention on compression friendliness and encryption efficiency, first encryption scheme can be adopted; while for applications putting more emphasis on the security of images, second encryption scheme can be used.

The rest of this paper is organized as follows: Section 2 introduces how we generate the plainimage-dependent secret key and the pseudorandom key-stream. Section 3 explains the implementation details our first encryption method. Section 4 presents the added RSV shuffling operation in the second encryption scheme. Detailed performance evaluations of these two encryption schemes are given in Section 5, and Section 6 gives a conclusion.

2 KEY PROCESSOR

To introduce diffusion property in the encryption process with respect to the plain-image, in both our encryption schemes, we first use the SHA-384 hashing function (YASUDA and SASAKI 2010) taking the plain-image as input to output a 384-bit random hash value, denoted as σ. Then we use Chen's chaotic system (Chen and Ueta 1999) to produce three pseudo-random key sequences, which can be described as follows:

$$\begin{cases} \dot{x} = a(y - x), \\ \dot{y} = (c - a)x - xz + cy, \\ \dot{z} = xy - bz. \end{cases} \tag{1}$$

where a, b and c are parameters which determine system's chaotic attractors and bifurcations. x, y and z make up the system's state, and x_0', y_0', and z_0' are the initial states. When $a = 35$, $b = 3$, $c \in [20, 28.4]$, the system is chaotic as shown in Figure 2.

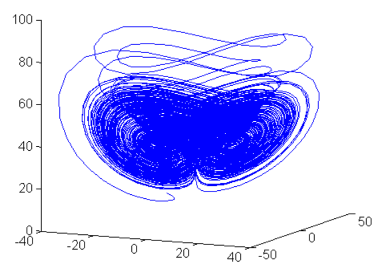


Fig. 2. Chen's chaotic system.

We convert the 384-bit hash value σ to 8-bit decimal form: $\sigma = k_1 k_2 \ldots k_{48}$, where each k_i ranged in $[0, 255]$. Then these 48 decimal numbers are used to modify the initial states of Equation (1) as follows,

$$\begin{aligned} x_0 &= x_0' + \frac{(k_1 \oplus k_2 \oplus \ldots k_{16})}{256}, \\ y_0 &= y_0' + \frac{(k_{17} \oplus k_2 \oplus \ldots k_{32})}{256}, \\ z_0 &= z_0' + \frac{(k_{33} \oplus k_2 \oplus \ldots k_{48})}{256}, \end{aligned} \tag{2}$$

and x_0, y_0, and z_0 are the new initial states of Equation (1). Iterate Chen's chaotic system for N_0 times, we set $N_0 = 10000$ in our experiment, for each iteration, three values x_i, y_i, and z_i can be obtained, $i = 1, 2, \ldots, 10000$. We pre-process these values using the following formula,

$$X_i = dec2bin(((abs(x_i) - floor(abs(x_i))) \times 10^{14}) \mod (M \times N)),$$
$$Y_i = dec2bin(((abs(y_i) - floor(abs(y_i))) \times 10^{14}) \mod (M \times N)),$$
$$Z_i = dec2bin(((abs(z_i) - floor(abs(z_i))) \times 10^{14}) \mod (M \times N)),$$

where function $dec2bin(x)$ converts decimal number x into binary number, $abs(x)$ returns the absolute value of x, $floor(x)$ rounds the element of x to the nearest integers less than or equal to x. M and N is the row number and column number of the plain-image. Therefore, in our proposed two schemes, the encryption key has two parts: 1) the 384-bit hash value σ; 2) the three initial states x_0', y_0', and z_0'. The three binary sequences X, Y, and Z are taken as the pseudo-random key-streams to control following encryption operations.

3 FIRST ENCRYPTION SCHEME

In this scheme, we realize encryption in JPEG's transformation stage and after quantization stage, which are denoted at positions (2) and (3) in Figure 1. We explain the implementation details of the in-transformation encryption and after-quantization encryption separately.

3.1 In-transformation Encryption

The modification we made for encryption in this part is to replace the original 8×8 DCT transformation by 16×16 DCT transformation. Hence, we initially raster scan the input plain-image to sequential 16×16 blocks, then apply 16×16 DCT for these blocks' transformation. In our encryption scheme, we still use JPEG's 8×8 quantization table, thus before the quantization procedure, the 256 coefficients of one 16×16 transformed block need to be distributed into four 8×8 blocks. And this work is controlled by the pseudo-random keystream X. For each transformed 16×16 block (denoted as $B16$), we distribute its 256 coefficients into four 8×8 blocks, represented by a 3-D array $B8$. In the coefficients distribution process, information denoting which one of the four 8×8 blocks possesses the 16×16 block's DC coefficient is saved as $DCIndex$, and this information will be used in the following after-quantization encryption. We explain the coefficients distribution process using Pseudo-codes 1.

In Pseudo-codes 1, i represents which 8×8 block is participated in this time's coefficients distribution, since there are four 8×8 blocks, 2 bits are needed from key-stream X each time. j indicates the number of coefficients that will be distributed into ith 8×8 block, and we use 3 bits from X to decide this number. The reason why we choose 3 bits is to avoid one 8×8 block possessing too many low frequency coefficients from the 16×16 block. After finishing all 16×16 blocks' coefficients distribution work, we quantize all 8×8 blocks using JPEG's quantization table, and do next step encryption.

3.2 After-quantization Encryption

In this part, we permute all quantized 8×8 blocks and confuse the DC coefficients in original 16×16 blocks through XOR operation using key-stream Y. The reason why we do not confuse DC coefficients in 8×8 block unit is because after 16×16 DCT transformation and 8×8 blocks' quantization, most DC coefficients in 8×8 blocks are zero. If we confuse the DC coefficients in all 8×8 blocks, then when inverse 16×16 DCT is performed on the decoder side for de-compression, many DC coefficients in 8×8 blocks will exceed the range of [0,255], which means that after the normalization process, the cipher-image will contain large scale of white/black pixels, a great amount of image information will be missing.

Pseudo-codes 1: Coefficients Distribution

```
Input: One 16×16 block B16, and keystream X
Output: Four 8×8 blocks (B8), and DC information (DCIndex)
CoeIndex ⟵ 1;
DCIndex ⟵ first 2 bits of X, and convert to decimal;
B8Number ⟵ 1 × 4 vector with four elements being zero;
while CoeIndex ≤ 256 do
    i ⟵ pick 2 bits from X, and convert to decimal;
    j ⟵ pick 3 bits from X, and convert to decimal;
    if B8Number(i) < 64 then
        Temp ⟵ B8Number(i) + j;
        if (Temp − 64) > 0 then
            j ⟵ j − (Temp − 64);
        end
        ith 8×8 block of B8 ⟵ pick j elements from B16 in zigzag scan order;
        B8Number(i) ⟵ B8Number(i) + j;
        CoeIndex ⟵ CoeIndex + j;
        Remove the first j elements from B16 in zigzag scan order;
        Remove the first 5 bits from X;
    else
        Remove the first 5 bits from X;
    end
end
return B8, DCIndex
```

The permutation method we use is the two key-driven cyclical shift (Zhou et al. 2014), where circular shift is operated on each column and each row separately, and the number of shift positions is controlled by Y. For a given input sequence, we first reshape it into a square matrix, then downward circular shift each column's element according to the shift position decided by the first half key-stream of Y. After shifting all columns, rightward circular shift each row's element according to the shift position decided by the second half key-stream of Y. We call the final produced permutation vector as P. In Figure 3, we give two examples to illustrate this permutation method. The two examples explain separately how we produce permutation vector when the length of input sequence S has square root or not. When using this permutation method for our 8×8 blocks' permutation, the input sequence S is the original order of all 8×8 blocks, and the output sequence S' is the permutation vector we need.

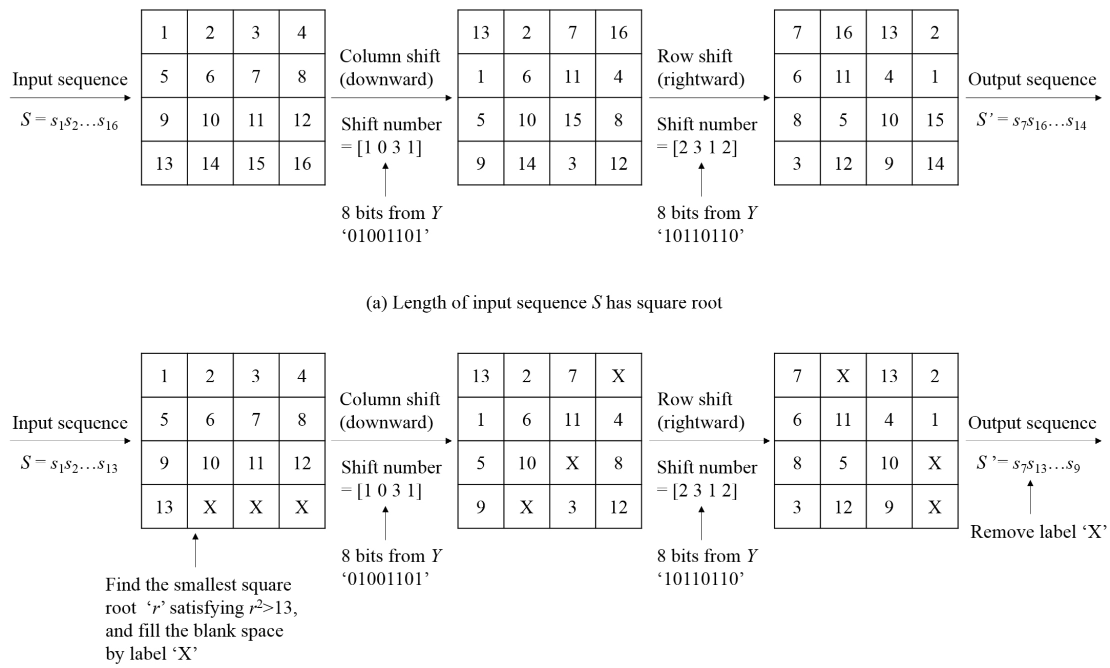
After performing block permutation, positions for original 16×16 blocks' DC coefficients (*DCIndex*) are changed. Finding the permuted *DCIndex* under the permutation vector S', we perform XOR operation on these permuted DC coefficients using Equation (3), to further improve the diffusion and confusion properties of our first encryption scheme.

$$dc_i = dc_i \oplus dc_{i-1} \oplus \cdots \oplus dc_1, \tag{3}$$

where dc_i is the DC coefficient of the ith permuted 8×8 block who has the original 16×16 block's DC coefficient, $i = 1, 2, \ldots, M \times N/256$. For recovering these confused DC coefficients, Equation (4) can be used:

$$dc_j = dc_j \oplus dc_{j-1}, \tag{4}$$

where j starts from $M \times N/256$, and ends with 2.



(a) Length of input sequence S has square root

(b) Length of input sequence S has no square root

Fig. 3. Two examples of circular shift.

3.3 Encryption and Decryption Algorithms for The First Scheme

Encryption algorithm of our first method mainly contains four parts: a) pseudo-random key-stream generation using SHA-384 and Chen's chaotic system; b) 16×16 DCT transformation, and coefficients distribution; c) block permutation and DC coefficients confusion; d) entropy encoding.

Encryption Algorithm-1

Step-1: Take plain-image as the input of SHA-384 algorithm to generate a 384-bit hash value, use this data to modify the three initial values (x_0', y_0', and z_0') by Equation (2), then run Chen's chaotic system for 10000 times to produce the pseudo-random key-streams X, Y, and Z;

Step-2: Level shift the plain-image by subtracting 128 from each pixel, and segment the level shifted image into non-overlapping 16×16 blocks. For each 16×16 image block, do

Step-2.1: Transform the block using 16×16 DCT;

Step-2.2: Distribute the 256 coefficients into four 8×8 blocks controlled by key-stream X, quantize these 8×8 blocks by JPEG's original order-8 quantization table;

Step-3: Repeat *Step-2* until all 8×8 blocks are quantized, permute these blocks using the two key-driven cyclical shift and key-stream Y, and do DC coefficients confusion according to Equation (3);

Step-4: Perform JPEG's entropy coding procedure for all processed 8×8 blocks, transmit the encrypted bit-stream, and encryption keys (384-bit hash value σ, three initial values x_0', y_0', and z_0') securely to decoder for decryption and decompression.

For decrypting the encrypted bit-stream, if encryption keys are not known, we just follow JPEG's decompression process to obtain the cipher-image, because of the format-compliant property. When keys are available, then the decryption algorithm is as follows:

Decryption Algorithm-1

Step-1: Use σ to modify x'_0, y'_0, and z'_0 by Equation (2), run Chen's chaotic system for 10000 times produce the pseudo-random key-streams X, Y, and Z;

Step-2: Perform entropy decoding procedure of JPEG, and generate the 8×8 blocks' permutation vector S' using the two key-driven cyclical shift and key-stream Y;

Step-3: Recover the confused DC coefficients using Equation (4), and then put the permuted 8×8 quantized blocks back to their original positions;

Step-4: For each four 8×8 blocks obtained in *Step-3*, do

Step-4.1: Perform JPEG's de-quantization process and coefficients re-distribution to construct the plain transformed 16×16 block according to key-stream X;

Step-4.2: Do inverse transformation using 16×16 DCT;

Step-5: Repeat *Step-4* until all 16×16 blocks are recovered, add 128 to all pixels to obtain the decrypted image.

---

4 SECOND ENCRYPTION SCHEME

In our first encryption scheme, we do not modify the entropy coding stage of JPEG, which means that to distinguish each 8×8 block, the end-of-block (EOB) identifiers are embedded into the zigzag-scan encoded sequence and this will bring a risk to encryption (Ji et al. 2015). In (Ji et al. 2015), they pointed out that these position seldom changed identifiers will leak the profile of the plain-image, correlation in 8×8 blocks cannot be efficiently removed. Hence they proposed to shuffle the positions of identifiers in the zigzag scan encoded sequence, to make the 8×8 pixel blocks different after encryption. Nevertheless, when shuffling these EOBs, they processed them independently, constraint that each block could only have 63 AC coefficients was not considered well in their cryptosystem, which may lead to the not format-compliant to JPEG problem, when the encryption key is not available and only decompression is performed.

In our second encryption method, we add an extra encryption step in JPEG's entropy coding stage on the basis of the first method. The major objective of this added encryption technique is to enhance the correlation removal ability, and meanwhile keep format compliance. Following the idea in (Ji et al. 2015), we change the position of EOBs by first shuffling the RSV pairs of AC coefficients, then embedding certain number of EOBs to keep format compliance. The shuffling operation is controlled by key-stream Z.

4.1 Shuffling RSV Pairs

In JPEG, after zigzag scanning quantized DCT coefficients, the DC and AC coefficients are entropy coded using Huffman coding separately. For DC coefficients, their differences are encoded, because DC coefficients of neighbouring blocks are highly correlated. While for AC coefficients, they are encoded using variable-length codes (VLC) in the form of RSV pairs. For example, given the following zigzag scan encoded sequence of AC coefficients in two blocks:

$$0, -2, 0, 1, -1, \underbrace{0, \ldots, 0}_{\text{58 zeros}}, 0, 0, 0, 5, \underbrace{0, \ldots, 0}_{\text{59 zeros}},$$

The resulting RSV pairs are

$$(1, -2), (1, 1), (0, -1), (0, eob), (3, 5), (0, eob).$$

where $(0, eob)$ is the stopping tag of one 8×8 block, which is essential for ensuring format compliance.

In the second encryption method, we first save all RSV pairs of the nonzero AC coefficients to form a list L^{ac}, the stopping tag $(0, eob)$ is not included in L^{ac}. To show where each 8×8 block ends, we count the number of nonzero AC coefficients in each block, denoted by $\zeta(i, j)$, for $1 \le i \le M/8$ and $1 \le j \le N/8$. Each element in $\zeta(i, j)$ is mapped to a unique RSV pair in JPEG's huffman table for AC coefficients, using the method proposed in (Ong et al. 2015). In the default table for AC coefficients, (Run/Size, Value)=(0, 1) and (0, -1) in category 1 are the shortest codewords, followed by (0, -3), (0, -2), (0, 2), and (0, 3) in category 2, and so forth. To maintain the final encrypted bit-stream size, we represent each of the possible 64 values (since $0 \le \zeta(i, j) \le 63$) by a codeword in category 1, 2, . . ., and 6. Specifically, we map (0, -1) to $\zeta(i, j) = 0$, (0, 1) to $\zeta(i, j) = 1$, (0, -3) to $\zeta(i, j) = 2$, (0, -2) to $\zeta(i, j) = 3$, and so on. We name these newly generated RSV pairs as L^{acNum}, which is used to distinguish each 8×8 block, and its length is $M \times N/64$. Concatenating L^{acNum} to the end of L^{ac} to form a new list L, we perform a permutation on L using key-stream Z to achieve AC coefficients encryption. The permutation vector is also produced through the two key-driven cyclical shift, and the permuted RSV pairs' list is denoted as L^p.

4.2 Embedding End-of-block Identifiers

Now we have obtained the permuted RSV pairs' list L^p, if the decoder knows key-stream Z, after JPEG's entropy decoding process, he/she just uses Z to produce the permutation vector to de-permute L^p, and gets L. Taking the final $M \times N/64$ RSV pairs to distinguish how many nonzero AC coefficients are included in each 8×8 block, he/she can easily put all nonzero AC coefficients' RSV pairs into their original 8×8 blocks. However, for someone who do not know the secret key, he/she cannot distinguish where each 8×8 block ends, then JPEG's decompression procedure may not be run successfully, which is the not format-compliant problem.

To ensure format-compliant when only JPEG de-compression process is performed, we embed $M \times N/64$ EOB identifiers into L^p. These embedded EOBs must satisfy three conditions: 1) the number of AC coefficients in each 8×8 blocks cannot exceed 63; 2) element before the embedded EOB must be nonzero AC coefficient, otherwise some zeros in L_p will be missing when the with-key decryption and decompression procedures are performed; 3) one EOB must be placed at the end of L_p. Our method is to embed the $M \times N/64$ EOBs one by one. For each embedding, we use two arrays: a forward-array and a backward-array, in which the forward-array is to ensure the currently embedded EOB satisfies the above three conditions, while the backward-array is to make sure that the remaining $(M \times N/64 - 1)$ EOBs can be successfully embedded into the left RSV pairs under the three constraints. Additionally, to evenly segment all RSV pairs of L_p into $\frac{M \times N}{64}$ 8×8 blocks, we use a poisson distribution with $\lambda = \frac{length\ of\ L_p}{(M \times N)/64}$ to produce a random number $Num_{avg} \sim P(\lambda)$. Then combining Num_{avg} with the previous two arrays, we can determine the specific position for the current EOB's embedding.

Pseudo-codes 2: End-of-block Identifiers Embedding

```
Input: Permuted RSV pairs (L_p),
    Maximum number of coefficients for 8×8 blocks (T),
    EOBs number (Num_eob)
Output: Encrypted AC sequence with EOBs (S_ac)
λ ← length(L_p) / ((M × N)/64);
T ← 63;
Num_eob ← (M × N) / 64;
while Num_eob ≥ 1 do
    Produce Num_avg ~ P(λ);
    Generate A_f and A_b from L_p;
    Index_closest ← index of element x in A_f satisfying x is the most closest element to Num_avg;
    Index_end ← max(find(A_f ≤ T));
    Index_start ← max(find(A_b > (Num_eob − 1) × T));
    if Index_closest > Index_end then
        Index_eob ← Index_end;
    else
        if Index_closest < Index_start then
            Index_eob ← Index_start;
        else
            Index_eob ← Index_closest;
        end
    end
    Put the first A_f(Index_eob) elements of L_p and one EOB identifier into S_ac;
    Num_eob ← Num_eob − 1;
    L_p ← remove the first A_f(Index_eob) elements from L_p;
end
return S_ac
```

We take the example in Section 4.1 to explain how we produce the forward-array (denoted as A_f) and the backward-array (denoted as A_b) for each EOB's embedding. Suppose the permuted RSV list L_p is

$$(1,-2),(1,1),(0,-1),(3,5)$$
$$\downarrow$$
$$0,-2,0,1,-1,0,0,0,5$$

No EOBs are included in this list. Create a new array A_{num} to count how many coefficients (including zero AC coefficients) are contained in each RSV pair, then A_{num} for the above example is

$$A_{num} = (2,2,1,4)$$

Generate the forward-array A_f by adding each element in A_{num} with its previous elements starting from A_{num}'s first element, while the backward-array A_b is produced by adding each element in A_{num} with its following elements starting from A_{num}'s last element. Then A_f and A_b will be

$$A_f = (2, 4, 5, 9),$$
$$A_b = (9, 7, 5, 4),$$

Specific realization of our EOB embedding method is given in Pseudo-codes 2.

4.3 Encryption and Decryption Algorithms for The Second Method

For our second encryption scheme, its encryption and decryption algorithms can be obtained by modifying *Step-4* in *Encryption Algorithm-1* and *Step-2* in *Decryption Algorithm-1*, respectively. Other steps remain the same.

Encryption Algorithm-2

Step-4: Use variable-length codes to represent AC coefficients in the form of RSV pairs, then

Step-4.1: Concatenate all RSV pairs to form a list L_{ac}, in which EOB identifiers are not included;

Step-4.2: Count the number of nonzero AC coefficients in each 8×8 block (ζ), and map these numbers to unique RSV pairs, according to the mapping table, to form another new list L^{acNum} with $M \times N/64$ elements;

Step-4.3: Add L^{acNum} to the end of L_{ac}, denoted as L, apply cyclical shift and key-stream Z to permuting L, and generate a new permuted RSV pairs' list L^p;

Step-4.4: Embed $M \times N/64$ number of EOB identifiers into L^p to obtain the final encrypted AC coefficients with end-of-block labels, denoted as S_{ac};

Step-4.5: Perform JPEG' entropy coding procedure for S_{ac} and DC coefficients, transmit the encrypted bit-stream and encryption keys (384-bit hash value σ, three initial values x'_0, y'_0, and z'_0) securely to decoder for decryption and decompression;

Decryption Algorithm-2

Step-2: Perform entropy decoding procedure of JPEG, discard all EOB labels, and obtain the permuted RSV pairs' list L^p, then

Step-2.1: Use key-stream Z and cyclical shift to de-permute L^p, and get L;

Step-2.2: Take the final $M \times N/64$ RSV pairs, and map them to the specific number of nonzero AC coefficients (ζ);

Step-2.3: Use ζ to put those remaining RSV pairs in L into their original permuted 8×8 blocks;

Step-2.4: Generate the 8×8 blocks' permutation vector S' using the two key-driven cyclical shift and key-stream Y;

For decryption without encryption keys, JPEG's decompression steps can be adopted directly on the encrypted bit-stream, because of the added EOB identifiers.

5 PERFORMANCE EVALUATION

In this section, we evaluate the performances of our proposed two encryption schemes, according to the four objectives of image encryption introduced in Section 1. In our experiment, parameters of Chen's system are: $a = 35$, $b = 3$, and $c = 28$. The three initial states are: $x'_0 = 10$, $y'_0 = -6$, and $z'_0 = 37$.

5.1 Encryption Security

Two kinds of security are included: perceptual security and cryptographic security. For perceptual security, we use PSNR value to evaluate cipher-image's distortion degree, and the results of the two encryption schemes are given in Figure 4. The tested plain-image is 'Baboon'. All images used in our schemes are taken from the USC-SIPI image database available on the website "http://sipi.usc.edu/database". It can be seen that when encryption keys are not known, *Algorithm-2* achieves a larger PSNR drop than *Algorithm-1*, which means a better image distortion ability. However, when the keys are available, the quality drop in the decrypted image of *Algorithm-2* is more serious than that of *Algorithm-1*, both compared with JPEG compressed image.

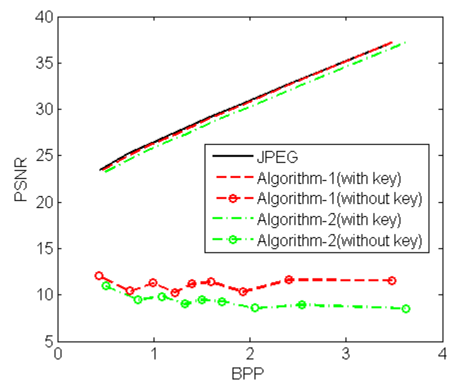


Fig. 4. Perceptual security results of two encryption schemes.

For the cryptographic security, we evaluate the performance of the two schemes under four cryptanalysis techniques: brute-force attack, differential attack, key sensitivity analysis, and statistical attack.

5.1.1 Brute-force attack. This attack is a typical attacking method in the ciphertext-only attack, in which only the encrypted data is available to attackers, and all cryptosystems should be designed to at least withstand this type of attack. Attackers try to recover the encrypted data through guessing all the possible keys. To make brute-force attack infeasible, the key space of a cryptosystem should be large enough. In both our two encryption schemes, the encryption keys are: 384-bit random hash value σ generated from SHA-384, three initial states of Chen's chaotic system x_0', y_0', and z_0'. For SHA-384 hash function, complexity of the best attacking method against this function is 2^{192} (YASUDA and SASAKI 2010). For the three initial values, if the precision is 10^{14}, the key space size is 10^{42}, thus the total key space size will be $2^{192} \times 10^{42}$, which is impossible for attackers to guess. Moreover, for each different plain-image, SHA-384 will produce a different 384-bit σ, and this also increases the attacking difficulty.

5.1.2 Differential attack. Differential attack is a chosen-plaintext attack, in which attacker is assumed to have the ability of modifying one pixel of the plain-image and observing the resulting cipher-image. By computing the difference between the chosen plain-images and the corresponding cipher-images, attacker tries to deduce a statistical relationship between them (Taneja et al. 2012). If such a minor change results in a significant change in the cipher-image, then the cryptosystem is considered as resistance against the differential attack. Two statistical evaluation parameters, net pixel change ratio (NPCR) and unified average change in intensity (UACI) (Wu et al. 2011), are widely used for checking the robustness of an image encryption scheme against differential attack. NPCR denotes the change rate of the number of pixels in the cipher-image while one pixel in plain-image is changed. The higher the value of NPCR, the more secure is the encryption scheme. UACI measures the average intensity difference between two cipher-images. To get higher security, the value of UACI should be close to 33%. The calculations of these two parameters are defined as

$$D(i,j) = \begin{cases} 0, if\ C^1(i,j) = C^2(i,j), \\ 1, if\ C^1(i,j) \neq C^2(i,j). \end{cases}$$

$$\text{NPCR} : N(C^1, C^2) = \frac{\sum_{i,j} D(i,j)}{M \times N} \times 100\%,$$

$$\text{UACI} : U(C^1, C^2) = \frac{\sum_{i,j} \frac{\left|C^1(i,j) - C^2(i,j)\right|}{255}}{M \times N} \times 100\%,$$

where C^1 and C^2 are two cipherimages corresponding to two plainimages differing by one single pixel.

We have computed the NPCR and UACI values for several images with their last pixel increased by 1, and the results for our two encryption algorithms are listed in Table 1. From Table 1, we can observe that *Algorithm-2* has better defence capability against differential attack than *Algorithm-1*, indicating a better diffusion property.

Table 1. Results of differential attack tests on different images

Image	*Algorithm-1*		*Algorithm-2*	
	NPCR%	UACI%	NPCR%	UACI%
Lena	0.9665	0.2289	0.9646	0.2868
Clock	0.9430	0.2287	0.9646	0.2744
Resolution chart	0.8719	0.2733	0.9406	0.3614
Chemical plant	0.9765	0.2292	0.9661	0.3305
Couple	0.9680	0.2228	0.9675	0.2827
Aerial	0.9778	0.2390	0.9685	0.3446
Stream and bridge	0.9812	0.2409	0.9661	0.3564
Peppers	0.9605	0.2081	0.9666	0.2638
Sailboat	0.9702	0.2238	0.9701	0.3017
Baboon	0.9702	0.2645	0.9505	0.3788

5.1.3 Key sensitivity analysis. An ideal cryptosystem should be extremely sensitive to the key used in the encryption/decryption algorithm, and it can be observed in two ways: (i) completely different cipherimages should be produced when slightly different encryption keys are used to encrypt the same plainimage; (ii) the cipherimage should not be correctly decrypted even if there is a minor difference in the encryption and decryption keys.

In our proposed two schemes, the encryption keys include two parts: 1) the 384-bit random hash value σ generated from SHA-384 and the plain-image; 2) three initial states x'_0, y'_0, and z'_0. In key sensitivity analysis, the plain-image is not changed, thus the 384-bit σ does not change, only the encryption/decryption results come from changes made in these three initial states need to be evaluated. To evaluate the first case of key sensitivity, we slightly change the three initial values $x'_0 = 10$, $y'_0 = -6$, and $z'_0 = 37$ into $x'_0 = 10.00000000000001$, $y'_0 = -6$, and $z'_0 = 37$, and use these two keys to encrypt 'Baboon' image under *Algorithm-1* and *Algorithm-2*. The encrypted two images (C1 and C2) by *Algorithm-1* and *Algorithm-2* using two different keys and their difference image are presented in Figure 5. For the second case of key sensitivity, if a minor change is occurred in encryption keys during transmission, in both decryption algorithms, the three pseudo-random key-streams X, Y, and Z will change, and this will consequently lead to wrong permutation vectors produced and wrong coefficients distribution results, thus the final decrypted cipher-image cannot be correct.

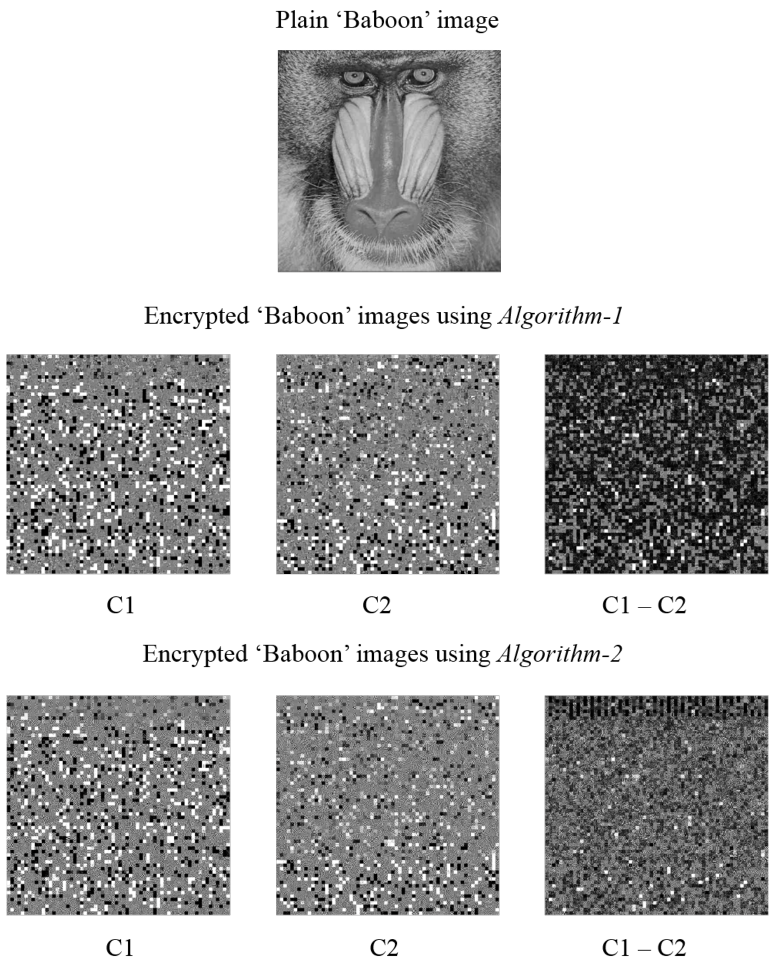


Fig. 5. Key sensitivity analysis for encryption process.

5.1.4 Statistical attack. In this type of attack, attackers try to predict the plain-image without the knowledge of key through studying the predictable relationship of some data segments between plain-image and cipher-image. It is obvious that most natural images have high correlation between adjacent elements, hence if an encryption scheme can reduce such correlation, the relationship between plain-image and cipher-image will be decreased, and the encryption scheme is deemed to be efficient. In Figure 6, we show the correlation distribution of two horizontally adjacent pixels, two vertically adjacent pixels and two diagonally adjacent pixels of the plain Baboon image and the encrypted images by *Algorithm-1* and *Algorithm-2*. From Figure 6, we can observe that correlation existed in *Algorithm-2* encrypted Baboon image are much less than that in plain Baboon image and *Algorithm-2* encrypted image, which illustrates the second encryption scheme has better decorrelation ability and superior defence ability against statistical attack, compared with the first encryption scheme.

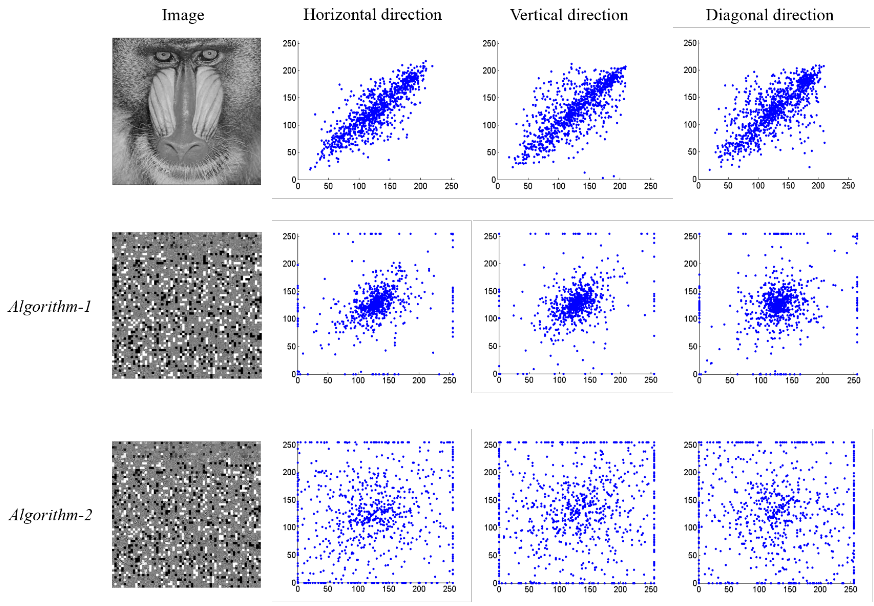


Fig. 6. Correlation charts of plain ‘Baboon’ image and encrypted image.

5.2 Format Compliance

In our first encrypted scheme, we only perform encryption in transformation stage and after quantization stage of JPEG, the entropy coding method for DC and AC coefficients remains the same, thus even without the encryption keys, decoder can still successfully recover the encrypted and compressed data using JPEG’s original de-compressor, and finally obtains the encrypted plain-image. In the second scheme, we shuffle all RSV pairs for AC coefficients and number of non-zero AC coefficients in each 8×8 block, if someone does not know the key to generate the shuffling vector, he/she cannot correctly recover the encrypted and compressed bit-stream data, because the side information of 8×8 blocks is missing. Hence, we propose to embed certain number of EOB identifiers to keep format compliance. Experiment results have confirmed that the embedded EOBs can ensure format-compliant when the key is not available to decoder.

5.3 Compression Friendliness

To evaluate the compression performance of the two schemes, we use bit-stream size (BS) and compression ratio (CR) as the evaluation criteria. The tested plainimage is ‘Baboon’, and the BS and CR values with different PSNR of JPEG, *Algorithm-1*, and *Algorithm-2* are given in Figure 7. We can observe from Figure 7 that, when PSNR is fixed, *Algorithm-1* has lower BS value and higher CR value than *Algorithm-2*. This is because in *Algorithm-2*, some overheads are introduced in the final encrypted bit-stream, and these overheads are the $M \times N/64$ EOB identifiers which are essential for the format compliant objective. Therefore, our second encryption scheme is less compression friendliness than the first encryption scheme.

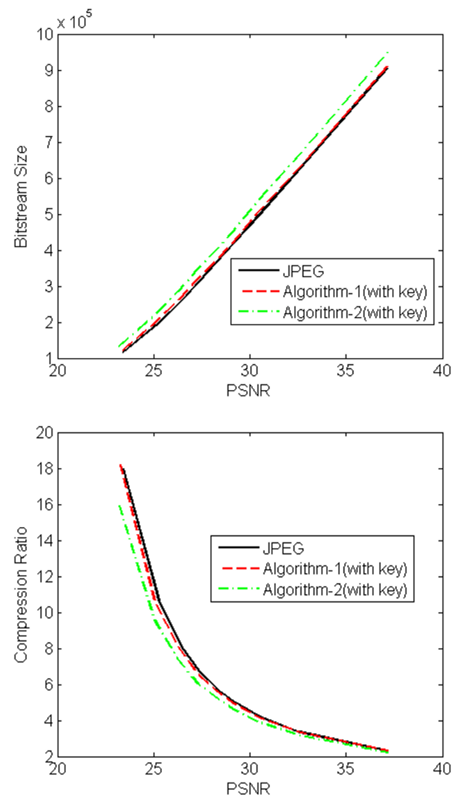


Fig. 7. Compression performance of two encryption schemes.

5.4 Encryption Efficiency

Apart from the security consideration, efficiency is also an important evaluation criterion for a good image cryptosystem, especially for real-time Internet application. In Table 2, we have listed the encryption speed of gray images with different sizes by using our proposed two encryption schemes and the well-known DES algorithm using electronic codebook (ECB) mode. The JPEG compression-only execution time is also given in the table as a reference. The simulation environment is MATLAB R2014a in 64 bit operating system, 3.50 Ghz, 16 GB RAM, Intel Core i7-4770K. From Table 2, we can see that the speed of our proposed two encryption schemes is much faster than the classic DES algorithm. However, *Algorithm-2*'s encryption speed is a little lower than *Algorithm-1*, and this is because in *Algorithm-2*, in order to enhance the security level and to keep format-compliant, we add the RSV shuffling and EOB labels embedding operations, which result in longer processing time.

Table 2. Efficiency comparison for different encryption schemes

Image size	Encryption time of *Algorithm-1* (s)	Encryption time of *Algorithm-2* (s)	Encryption time of DES (s)	JPEG compression (s)
256×256	1.41	1.74	33.49	0.81
512×512	5.83	7.93	240.56	1.54
1024×1024	23.89	36.97	2634.91	5.53

6 CONCLUSIONS

This paper presents two image encryption schemes based on 16×16 DCT, which are realized at the intermediate stages of JPEG. They both have their own advantages and disadvantages. For the first encryption scheme, it is more compression friendliness and needs less execution time, but its security level are not so high, hence it may suitable for applications where real-time interaction is more important than confidentiality. While for the second scheme, it obtains a higher protection ability against the differential attack and statistical attack compared with the first scheme, at the price of sacrificing compression performance and encryption efficiency.

In both of our encryption schemes, we use adaptive key to control the whole encryption techniques, and the key is different for each different plain-image. For the first encryption scheme, we realize encryption in transformation stage and after quantization stage of JPEG, which includes the 16×16 DCT transformation, 8×8 block permutation and DC coefficients confusion. For the second scheme, after finishing all encryption operations in the first scheme, we add the entropy coding stage encryption to enhance the security level by shuffling all RSV pairs of nonzero AC coefficients, according to the secret encryption key. Finally, to maintain the format-compliant property, EOB identifiers denoting the side information of each 8×8 block are embedded into these shuffled RSV pairs.

REFERENCES

Guanrong Chen and Tetsushi Ueta. 1999. Yet another chaotic attractor. *International Journal of Bifurcation and chaos* 9, 07 (1999), 1465–1466.

Osama S Faragallah. 2015. Efficient confusion–diffusion chaotic image cryptosystem using enhanced standard map. *Signal, Image and Video Processing* 9, 8 (2015), 1917–1926.

Sesha Pallavi Indrakanti and PS Avadhani. 2011. Permutation based image encryption technique. *International Journal of Computer Applications (0975–8887) Volume* (2011).

Snehashis Jha. 2014. *Image compresssion and encryption using scan pattern.* Ph.D. Dissertation.

Xiao-yong Ji, Sen Bai, Yu Guo, and Hui Guo. 2015. A new security solution to JPEG using hyper-chaotic system and modified zigzag scan coding. *Communications in Nonlinear Science and Numerical Simulation* 22, 1 (2015), 321–333.

Manish Kumar, Pradeep Powduri, and Avinash Reddy. 2015. An RGB image encryption using diffusion process associated with chaotic map. *Journal of Information Security and Applications* 21 (2015), 20–30.

Shiguo Lian. 2008. *Multimedia content encryption: techniques and applications.* CRC press.

Suchindran S Maniccam and Nikolaos G Bourbakis. 2004. Image and video encryption using SCAN patterns. *Pattern Recognition* 37, 4 (2004), 725–737.

Ayoub Massoudi, Frédéric Lefebvre, Christophe De Vleeschouwer, Benoit Macq, and J-J Quisquater. 2008. Overview on selective encryption of image and video: challenges and perspectives. *EURASIP Journal on Information Security* 2008, 1 (2008), 1.

A Mitra, YV Subba Rao, SRM Prasanna, and others. 2006. A new image encryption approach using combinational permutation techniques. *International Journal of Computer Science* 1, 2 (2006), 127–131.

SimYing Ong, KokSheik Wong, Xiaojun Qi, and Kiyoshi Tanaka. 2015. Beyond format-compliant encryption for JPEG image. *Signal Processing: Image Communication* 31 (2015), 47–60.

Devaraj Ponnain and Kavitha Chandranbabu. 2016. Crypt analysis of an image encryption algorithm and an enhanced scheme. *Optik-International Journal for Light and Electron Optics* 127, 1 (2016), 192–199.

Zhenxing Qian, Xinpeng Zhang, and Shuozhong Wang. 2014. Reversible data hiding in encrypted JPEG bitstream. *IEEE Transactions on Multimedia* 16, 5 (2014), 1486–1491.

Nidhi Taneja, Balasubramanian Raman, and Indra Gupta. 2012. Combinational domain encryption for still visual data. *Multimedia tools and applications* 59, 3 (2012), 775–793.

Zhou Wang and Alan C Bovik. 2002. A universal image quality index. *Signal Processing Letters, IEEE* 9, 3 (2002), 81–84.

Chung-Ping Wu and C-CJ Kuo. 2005. Design of integrated multimedia compression and encryption systems. *IEEE Transactions on Multimedia* 7, 5 (2005), 828–839.

Yue Wu, Joseph P Noonan, and Sos Agaian. 2011. NPCR and UACI randomness tests for image encryption. *Cyber journals: multidisciplinary journals in science and technology, Journal of Selected Areas in Telecommunications (JSAT)* (2011), 31–38.

Yanyan Xu, Lizhi Xiong, Zhengquan Xu, and Shaoming Pan. 2014. A content security protection scheme in JPEG compressed domain. *Journal of Visual Communication and Image Representation* 25, 5 (2014), 805–813.

Kan YASUDA and Yu SASAKI. 2010. Cryptographic Hash Functions:. *IEICE ESS Fundamentals Review* 4, 1 (2010), 57–67. DOI: https://doi.org/10.1587/essfr.4.57

Siu-Kei Au Yeung, Shuyuan Zhu, and Bing Zeng. 2009. Partial video encryption based on alternating transforms. *Signal Processing Letters, IEEE* 16, 10 (2009), 893–896.

Siu-Kei Au Yeung, Shuyuan Zhu, and Bing Zeng. 2011a. Design of new unitary transforms for perceptual video encryption. *Circuits and Systems for Video Technology, IEEE Transactions on* 21, 9 (2011), 1341–1345.

Siu-Kei Au Yeung, Shuyuan Zhu, and Bing Zeng. 2011b. Perceptual video encryption using multiple 8× 8 transforms in H. 264 and MPEG-4. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on.* IEEE, 2436–2439.

Mohammad Ali Bani Younes and Aman Jantan. 2008. An image encryption approach using a combination of permutation technique followed by encryption. *International journal of computer science and network security* 8, 4 (2008), 191–197.

Bing Zeng, Siu-Kei Au Yeung, Shuyuan Zhu, and Moncef Gabbouj. 2014. Perceptual encryption of H. 264 videos: Embedding sign-flips into the integer-based transforms. *Information Forensics and Security, IEEE Transactions on* 9, 2 (2014), 309–320.

Dinghui Zhang and Fengdeng Zhang. 2014. Chaotic encryption and decryption of JPEG image. *Optik-International Journal for Light and Electron Optics* 125, 2 (2014), 717–720.

Jiantao Zhou, Xianming Liu, Oscar C Au, and Yuan Yan Tang. 2014. Designing an efficient image encryption-then-compression system via prediction error clustering and random permutation. *IEEE transactions on information forensics and security* 9, 1 (2014), 39–50.