Image compression and encryption scheme using fractal dictionary and Julia set

Yuanyuan Sun¹, Rudan Xu¹, Lina Chen², Xiaopeng Hu¹

¹College of Computer Science and Technology, DaLian University of Technology, Dalian, People’s Republic of China
²National Astronomical Observatories, Chinese Academy of Sciences, Beijing, People’s Republic of China
E-mail: syuan@dlut.edu.cn

Abstract: An efficient and secure environment is necessary for data transmission and storage, especially for large-column multimedia data. In this study, a novel compression–encryption scheme is presented using a fractal dictionary and Julia set. For the compression in this scheme, the fractal dictionary encoding not only reduces time consumption, but also gives good quality image reconstruction. For the encryption in the scheme, the key has large key space and high sensitivity, even to tiny perturbation. Besides, the stream cipher encryption and the diffusion process adopted in this study help spread perturbation in the plaintext, achieving high plain sensitivity and giving an effective resistance to chosen-plaintext attacks.

1 Introduction

An efficient and secure environment is necessary for data transmission and storage, especially for multimedia that involves audio, image or video. In recent years, much research on multimedia security has been widely discussed, which can be classified into two types – raw image encryption and compressed image encryption. In the raw image encryption, the common approach is based on scrambling techniques, using permutation or affine transformation. However, because of the scrambling-only scheme, the encryption is most likely to be broken. On the other hand, taking large-volume data of multimedia and high encrypting time consumption into consideration, it becomes more reasonable to encrypt the compressed data.

In multimedia applications, compression–encryption algorithms are usually designed to recover the approximate value of the plaintext data stream. As for multimedia data, an approximation of the original multimedia content is sufficient in most applications as long as error between the original and the recovered data is small [1]. In compressed image encryption, the image is compressed before encryption. Recently, chaos-based or fractal-based ciphers are often used in compressed image encryption because both chaos and fractals sets cost less storage space and have input-sensitivity. For example, Yuen and Wong [2] combined the chaotic tent map with discrete cosine transformation (DCT) and secure hash algorithm-1 (SHA-1). Hermassi et al. [3] proposed a method of joint compression and encryption using chaotically mutated Huffman trees. In fractal-based compressed image encryption, Bourbakis [4] presented a novel image compression–encryption methodology by using fractals. Lian et al. [5] first proposed a solution for image encryption in fractal coding. Lock et al. [6] converted a Mandelbrot set and compressed image into square matrix before encrypting images.

In this paper, we utilise a Julia set as stream cipher to encrypt a compressed image base on fractal dictionary encoding. The remainder of this paper is organised as follows: Section 2 gives a brief overview of Julia set and fractal dictionary; the proposed compression–encryption method is described in Section 3; the experimental results are reported in Section 4, which is followed by the conclusions.

2 Overview of Julia set and fractal dictionary

2.1 Julia set

Iterated function systems are introduced as a unified way of generating a broad class of fractals that contains classical Cantor sets, Sierpinski gaskets, Julia sets and much more. Many of these sets are traditionally viewed as being produced by a process of successive microscopic refinement taken to the limit [7].

The classical Julia set is a product of several iterations by a mapping function \( f(z) = z^2 + c \). Since the complex number \( z = x + iy \) can be uniquely mapped to \((x, y)\) in real number space, a one-to-one relationship can be established between the complex number space and the real number space. In practice, we select an area in complex plane and map it to a screen area \( W \) on the computer screen where one pixel is related to a point \( z_0 \) in Julia set. In this paper, the Julia set is constructed by the time-escaped algorithm, which is described as follows.

First, initialise the screen area \( W \), the escaped radius ‘Rad’ and the elapsed time limit ‘LimitTime’ where \( W \) contains \( n \times m \) pixels. ‘Rad’ \( \geq \max(2, |c|) \) and ‘LimitTime’ is the number of the maximum iteration.

Secondly, initialise the complex number \( z_0 = x_0 + iy_0 \) and iterate with the mapping function \( z = z^2 + c, p, q, \ldots \)
\( q \in R \). \((x_0, y_0)\) corresponds to each point in \( W \) before iteration. There are \( n \times m \) pixels in \( W \) which are presented as (1), where \((0, 1, \ldots, n-1)\) are the \( X \) or \( Y \) coordinates. The details are as follows

\[
\begin{align*}
\text{Step 1: } & \text{Initialise } j=0 \ (j \in (0, \text{LimitTime})) \text{ and the Julia parameters } p \text{ and } q \text{ in } c=p+qj. \\
\text{Step 2: } & \text{Calculate } z_j = z_{j-1}^2 + c, \text{ where } z_j = x_j + iy_j, \ (x_j, y_j) \text{ represents the location of initial point } z_0 \text{ after } j \text{th iteration. Set } j=j+1 \text{ after calculating.} \\
\text{Step 3: } & \text{Calculate } r = |z_j| = \sqrt{x_j^2+y_j^2}.
\end{align*}
\]

If \( r > '\text{Rad}' \), choose a unique colour for the point \((x_0, y_0)\) according to \( j \), representing this point is an escaped point, and go to step 4; If \( r <= '\text{Rad}' \) and \( j = '\text{LimitTime}' \), assign a predefined and shared colour to the point \((x_0, y_0)\) meaning that this point is converged and does not escape within ‘LimitTime’, and go to step 4; and If \( r < '\text{Rad}' \) and \( j < '\text{LimitTime}' \), go to step 2.

\text{Step 4: Choose next point in } W \text{ and assign it to } (x_0, y_0), \text{ then go to step 2.}

For example, we started with the first point \((x_0, y_0)\) at \((0, 0)\). After the first iteration, \((x_0, y_0)\) changes to \((x_1, y_1)\). After the \(j\)th iteration, \((x_{j-1}, y_{j-1})\) changes to \((x_j, y_j)\). When the calculation on the first pixel \((0, 0)\) is completed, we deal with the next point \((x_0, y_0)\) at \((0, 1)\) the same way as the previous one. Repeat the above steps until all the pixels in \( W \) have been calculated.

If \( r > '\text{Rad}' \) and \( j = 10 < '\text{LimitTime}' \), meaning that this point escapes at the tenth iteration, then we mark this point as \((j \mod 255 = 10 \mod 255 = 10)\), and go to step 4; if \( r <= '\text{Rad}' \) and \( j = '\text{LimitTime}' \), meaning that it does not escape within the ‘LimitTime’ iterations, then assign the colour 255 to this point and go to step 4.

Fig. 1 is the classical Julia sets with different \( c = p + iq \) generated by the time-escaped algorithm. Actually, it has been proved that the points closer to the Julia set border have a more pronounced border’s features, that is, closed set, density, non-differentiability and sensitivity to initial conditions. Furthermore, Julia set is more accurate if the iteration ‘LimitTime’ is bigger, presenting a more detailed Julia set border. In our experiments, ‘LimitTime’ = 1000 and ‘Rad’ = 10.

### 2.2 Block truncation coding

Delp and Mitchell [8] presented a block truncation coding (BTC) scheme for image compression. It is a type of lossy image compression technique for grey-scale images. In this method, each block can be converted to a BTC value by using (2) where \( x \) is the pixel of the block and \( x_{\text{ave}} \) is its average value

\[
\tilde{x} = \begin{cases} 
1, & x \geq x_{\text{ave}} \\
0, & x < x_{\text{ave}}
\end{cases}
\]

Using (2), we can obtain a matrix with each element \( \tilde{x} \) marked as 0 or 1. Regarding this matrix as a vector with a binary sequence, the corresponding decimal number can be calculated which is called BTC value.

### 2.3 Fractal dictionary

In the traditional fractal encoding, devised by Barnsley, an image is first partitioned into non-overlapping cells as range blocks and then the image is divided into overlapping sub-blocks, which are named as domain blocks. The image is titled by range blocks, each of which is mapped from one of the domain blocks as depicted in Fig. 2. The combined mappings constitute a transform on the image as a whole [9]. In the scheme, an image can be encoded by the transform, which is generated by collage theorem. The collage theorem characterises an iterated function system whose attractor is close to a given set.

Suppose that the domain block size is \( B \times B \) and the range block size is \( R \times R \) for the image of size \( N \times N \). As Fig. 2 shows, there are \((N-B+1) \times (N-B+1)\) overlapping domain blocks in the domain block pool and \((N/R) \times (N/R)\) non-overlapping range blocks in range block pool. The traditional scheme utilises self-similarity in image and tries to find the most similar domain block that minimises the distortion (see (3)) in the domain block pool for each range block, recoding the corresponding transform. The transform is represented by specifying the identity of the matching domain block together with the block mapping parameters [9]. For a \( 256 \times 256 \) image, there are 62 001 domain blocks when the size of domain block is \( 8 \times 8 \). For each range block, searching the best matching domain block needs 62 001 \times 8 \) comparisons because there are eight kinds of affine transformations for each domain block (see Section 3.1). It is very time consuming, especially for images with large size. Therefore the most important thing on the fractal coding research is to reduce the coding time without loss of image quality.

![Fig. 1 Julia sets generated by time-escaped algorithm](image.png)

\[
p=0.8275, \ q=0.1725 \quad p=-0.3, \ q=0.71 \quad p=-0.75, \ q=0
\]
Recently, scholars also paid attention to improving fractal encoding time [10]. The authors combined the block classification and sorting based on Pearson’s correlation coefficient, accelerating the encoding time. Besides, Jaferzadeh et al. [11] used a fuzzy c-mean-clustering approach and compared with discrete-cosine-transform-based metric to relieve the computational burden. In this paper, as Section 2.1 stated, the Julia set has rich contents and its border is disordered and complicated. With these properties, Sun and Kong [12] present an image compression approach based on \( M - J \) set. A series of Julia sets are generated by a Mandelbrot set, each of which is partitioned into domain blocks with a BTC classification and written into a shared dictionary. Since each domain block owns a unique BTC value and a BTC value can be shared by a series of domain blocks, the BTC value can be treated as a classifier for the domain blocks. The domain blocks sharing a same BTC constitute a BTC queue, shown in Fig. 3.

The \( M \) blocks having big distortion between each other are selected as a BTC queue. In this way, the encoding process is simplified to calculate the range block’s BTC value and find the best matching domain block in the corresponding BTC queue, rather than in the self-domain block pool of the image.

In the case of \( N = 256, B = 8, R = 4 \) and \( M = 10 \), the traditional method searches the best matching domain block by \( 62 \, 001 \times 8 \) comparisons for each range block and \( 62 \, 001 \times 4096 \times 8 \) comparisons for the whole image. However, the scheme in [12] only needs \( 10 \times 8 \) comparisons for each range block, \( 10 \times 4096 \times 8 \) comparisons in total. It is obvious that the encoding time apparently decreases.

3 Proposed method

In this paper, an image compression–encryption scheme is presented. The fractal dictionary-based encoding is adopted in this scheme to overcome the high encoding computation burden. Then, the Julia set is converted to a stream cipher and encoded with the compressed data before diffusion process.

3.1 Compression and decompression

For fractal encoding based on fractal dictionary, an image is first partitioned into \((N/R) \times (N/R)\) non-overlapping range blocks covering the whole image. Each range block should be compared with \( M \) domain blocks in the corresponding
BTC queue and the best matching domain block with the minimum distortion is recorded together with the encoding parameters, which will be discussed later. Usually, distortion criterion, defined as (3), is used to measure the distortion between the range block and affine transformed domain block

\[ d(X, D) = \sum_{i,j=1}^{N} (x_{ij} \times s + o - y_{ij}) \tag{3} \]

where \( X \) is the range block, \( D \) is the transformed domain block and \( x_{ij} \) and \( y_{ij} \) are the pixel values at \((i,j)\) coordinates of \( X \) and \( D \), respectively. The transform minimising the distortion criterion is recorded, consisting of contrast scalar \( s \) (see (4)), luminance offset \( o \) (see (5)) and affine transformation \( t \) together with the BTC value and the position of the best matching domain block ‘offset’. Fig. 4 shows the format of one record. The affine transformation \( t \) is one of eight affine transformations shown in Table 1

\[ s = \frac{N \sum_{j=1}^{N} x_{ij}y_{ij} - \sum_{j=1}^{N} x_{ij} \sum_{j=1}^{N} y_{ij}}{N \sum_{j=1}^{N} x_{ij}^2 - \left( \sum_{j=1}^{N} y_{ij} \right)^2} \tag{4} \]

\[ o = \frac{1}{N} \left[ \sum_{j=1}^{N} x_{ij} - s \sum_{j=1}^{N} y_{ij} \right] \tag{5} \]

The fractal encoding is a lossy compression, which means that a minor perturbation in image may have no changes in the corresponding compression codes. However, a secure encryption system should be sensitive to a minor change in image, that is, it should have high plaintext sensitivity. To ensure the sensitivity to the encoding parameter \( s \) as well as \( o \), larger storage spaces for \( s \) and \( o \) work better. We utilised the data without being quantified when conducting compression with \( s \) and \( o \). The experiments with each pixel least significant bit (LSB) changed in the 256 × 256 Aeroplane image show that a tiny change in image causes the changes in the encoding record, confirming the plaintext sensitivity.

For fractal decoding based on fractal dictionary, the image restoration uses the BTC value and the ‘offset’ to locate the best matching domain block. Applying (6), we obtain a reconstruction for one range block. As all range blocks cover the whole image, the final decoded image is restored

\[ R_i = s_i \times D_i + o_i \times U \tag{6} \]

Table 1 Eight kinds of affine transformations [13]

| Transformation ID | Action                                      |
|-------------------|---------------------------------------------|
| \( f_0 \)         | identity                                    |
| \( f_1 \)         | orthogonal reflection about mid-vertical axis of block |
| \( f_2 \)         | orthogonal reflection about mid-horizontal axis of block |
| \( f_3 \)         | orthogonal reflection about first diagonal of block |
| \( f_4 \)         | orthogonal reflection about second diagonal of block |
| \( f_5 \)         | rotation about centre of block, through +90° |
| \( f_6 \)         | rotation about centre of block, through +180° |
| \( f_7 \)         | rotation about centre of block, through −90° |

Note that \( R_i \) is the recovered range block, \( D_i \) is the transformed domain block in the dictionary with the \( n \)th affine transformation and \( U \) is a matrix with the same size as the \( D_i \), whose elements are all 1. As (6) shows, the image reconstruction only needs one operation less than eight iterations in the traditional way. Therefore it works better in recovering time consumption as well as iteration error.

3.2 Data preprocessing

In this scheme, the compressed data are preprocessed before encryption and the total bit allocation for one recode is assigned to \( 8 \times n \ (n = 1, 2, 3, \ldots) \) by default. For BTC ranging from 0 to 65 535, we assign it 16 bits of storage space. For the offset of best matching domain block in the BTC queue, we assign it 5 bits. There are many allocation cases for \( s \) and \( o \), each of which results in different compression ratio. The compression ratio is defined as (number of bits in the image)/(number of bits in the compressed image). As Section 3.1 discussed, we carried out another experiment shown in Table 2. Table 2 shows the sensitivity of some cases of bit allocation for \( s \) and \( o \). In this paper, for \( s \) and \( o \), we assign them 12 and 20 bits, respectively, to make these two parameters sensitive to the change of LSB. The kind of affine transformation is assigned to 3 bits. In this case, the compression ratio is (4 ×
4 \times 8)/56 = 2.286. The detail is shown in Fig. 4. After data preprocessing, we obtain a temporary plaintext.

### 3.3 Encryption and decryption

Stream ciphers can be viewed as approximating the action of a proven unbreakable cipher, the one-time pad [14]. The keystream is combined with one plaintext digit at a time to form the ciphertext and is typically generated serially from a random seed value using digital shift registers. It makes use of a much smaller and more convenient key such as 128 bits. Based on this key, it generates a pseudorandom keystream and the keystream can be combined with the plaintext digits in a similar fashion to the one-time pad. Therefore it has an advantage of plaintext hidden, which is good to resist chosen-plaintext attack and known plaintext attack. For the beginning of the encryption, it prefers an initial stream cipher with inconspicuous regulation and no statistical features.

The classical Julia set (shown in Fig. 5a) is a product by mapping function \( f(z) = z^2 + c \). Furthermore, when the superscript of \( z \) is not 2, that is, when the \( m \) in mapping function \( f(c) = z^m + c \) is not equal to 2, the product by this mapping function is called the generalised Julia set (shown in Fig. 5b). In this paper, the generalised Julia set is conducted by the time-escaped algorithm, depicted in Section 2.1.

Fig. 5 shows that the generalised Julia set has a much richer content than the classical Julia set. We select some locations of the generalised Julia set, enlarging them into 256 × 256 image as shown in Fig. 6.

From Fig. 6, we see that the Julia set changes a lot with the location translocating. Due to the fact that the generalised Julia set has a rich content, inconspicuous regulation feature and its inner set changes according to its location, we select an inner set of the generalised Julia set as initial stream cipher. Therefore the encryption keys for Julia generation are the parameters of the mapping function \( f(z) = z^m + c (c = p + q i) \) and the positions of the start locations \( X, Y \), five keys in total.

In this paper, the inner Julia set of the size \( K \times K \) is used to initialise the stream cipher. The pixels in the Julia set are all converted to 8 bits, forming a \( K \times K \times 8 \) length stream cipher denoted by JuliaStreamCipher. The stream cipher moves 1 bit to left after its seed is made the exclusive OR operation (XOR) with a temporary plaintext digit. The last digit of the stream cipher is filled with a new bit by XOR on the current ciphertext with two seeds. The ciphertext generation is shown in (7) and the keystream generation is shown in (8). After the encryption, we obtain temporary ciphertext

\[
C_i = Z_i^0 \oplus Z_i^1 \oplus \text{Plain}_i, \quad (0 \leq i < \text{plainlength}) \quad (7)
\]

\[
Z_j^i = \begin{cases} 
(Z_{j-1}^i, Z_{j-2}^i, \ldots, Z_{j-2}^i) \ll 1, & 0 \leq j < \text{stream length} - 1 \\
Z_{50}^i \oplus Z_{100}^i \oplus C_j, & j = \text{stream length} - 1 \\
1 \leq i < \text{plain length}
\end{cases} 
\]

\[
Z^0 = \text{Julia Stream Cipher}_j, \quad 0 \leq j < \text{stream length} \quad (8)
\]

Note that \( Z_j^i \) represents a keystream digit in the \( j \)th position at the \( i \)th shift, \( Z^0 \) represents the initial stream cipher ‘JuliaStreamCipher’, \( C_j \) represents a ciphertext digit at the \( i \)th position, ‘plainlength’ is the bit stream length of plaintext and ‘streamlength’ is the length of stream cipher \((K \times K \times 8)\). Equations (7) and (8) show that a single-error in keystream can affect the rest of the keystream, causing the ciphertext to change correspondingly. Therefore the stream cipher encryption helps spread the perturbation of plaintext or keystream and realise diffusion to a certain extent.

Decryption is the reverse order of encryption which is shown in (9) and (10)

\[
\text{Plain}_i = Z_i^2 \oplus Z_{100}^i \oplus C_i (0 \leq i < \text{plain length}) \quad (9)
\]
(X,Y): (76, 119)  (X,Y): (104, 68)  (X,Y): (117, 194)  (X,Y): (192, 149)

Fig. 6  Julia sets at different locations

$$Z_j = \begin{cases} (Z_{0j}^{-1}, Z_{1j}^{-1}, ..., Z_{255j}^{-1}) \ll 1, & 0 \leq j < \text{stream length} - 1 \\ Z_{0j} \oplus Z_{100j} \oplus C_j, & j = \text{stream length} - 1 \\ (1 \leq i < \text{plain length}) \end{cases}$$

$$Z^0 = \text{Julia Stream Cipher}, \quad 0 \leq j < \text{stream length}$$  \hspace{1cm} (10)

3.4 Diffusion and de-diffusion

For a secure encryption scheme, a mechanism of diffusion is necessary; otherwise the opponent can break the cryptosystem by comparing a pair of plaintext and ciphertext to discover some useful information [15]. This mechanism improves the ability of anti-jamming. It also significantly changes the statistical properties of the diffusing data by spreading the influence of each bit of the data on the whole the cipher-image.

To enhance the diffusion performance, the current ciphertext $q_i$ is influenced by the current temporary ciphertext $p_i$, the next temporary ciphertext $p_{i+1}$ and the previous two ciphertexts $q_{i-1}$ and $q_{i-2}$. The whole diffusion process needs two keys: key1 and key2. The process is depicted as (11), where ‘length’ is the temporary ciphertext length whose value is equal to ‘plainlength’ (see (11))

The de-diffusion is the reverse process of the diffusion, as shown in (12) (see (12))

4 Experimental results

We selected some images in the Miscellaneous [16] as our experimental material. All the experiments were conducted on a Core(TM) i5(2.40 GHz) personal computer. The default parameter of range block in compression process is $R = 4$. For encryption, the default parameters of Julia mapping function are $m = 15$, $p = 0.5$, $q = -0.7$, $X = 205$ and $Y = 34$. The diffusion keys are key1 = 13 and key2 = 29.

4.1 Compression performance analysis

On one hand, time consuming is a big challenge for conventional fractal encoding. Usually, it costs 45 s to encrypt an image of the size $256 \times 256$. On the other hand, because the fractal encoding is a lossy compression, trying to make the reconstructed image as close to the original image as possible is another factor to be taken into consideration. Peak signal-to-noise ratio (PSNR) is commonly used to measure the quality of reconstruction of lossy compression algorithms, which is defined as (13). Typical values for the PSNR in lossy image and video compression are between 30 and 50 dB, provided the bit depth is 8 Bit, where higher is better

$$\text{PSNR} = 10 \log \frac{\text{max}^2}{\text{MSE}} \hspace{1cm} (13)$$

Note that MSE is mean squared error between two images, and ‘max’ is the maximum possible pixel value of the image. In this case, max = 255.

In the compression system, the whole compression time includes two parts – the data preprocessing and the compression. As Fig. 7 demonstrates, the compression time of boat image is 949 ms, which is a satisfying result. At the same time, the restored image is close to the original image by human visual with 32.3705 PSNR. Table 3 shows the times of compression/decompression and the PSNRs of three $256 \times 256$ images.

It is easy to conclude that the compression based on fractal dictionary has a good performance on the time consumption as well as the image reconstruction. Besides, the time consumption for image restoring is very low. One iteration for image reconstruction, as discussed in Section 2.3, can explain it.

4.2 Key space analysis

A large storage space of secret keys can be difficult for attackers to break, thus protecting its encryption security. In this scheme, the secret keys include stream cipher keys and diffusion keys, seven keys in total, as shown in Table 4.

It is seen from Table 4 that the key space is $2^{32} \times 2^{264} \times 3 \times (256 - K + 1) \times (256 - K + 1) = (256 - K + 1) \times (256 - K + 1) \times 2^{256}$. If the attacker employs a 1000 MIPS computer to guess the key by brute force, the computational
For plaintext sensitivity, if one bit of plaintext changed, the corresponding ciphertext should also be much different. The different pixels percentage between the two ciphertexts is an index of key sensitivity. The common standards to test plain image sensitivity are the number of pixels change rate (NPCR) and uniform average changing intensity (UACI), defined as (14) and (15):

\[
\text{NPCR} = \frac{1}{W \times H} \sum_{i=1,j=1}^{W,H} D(i, j) \times 100\% \quad (14)
\]

\[
\text{UACI} = \frac{1}{W \times H} \sum_{i=1,j=1}^{W,H} \left| C_1(i, j) - C_2(i, j) \right| \times 100\% \times \frac{1}{255} \quad (15)
\]

where \( C_1(i, j) \) is a digit in one ciphertext and \( C_2(i, j) \) is the digit in the other one. \( D(i, j) \) in (14) is determined by \( C_1(i, j) \) and \( C_2(i, j) \) namely, if \( C_1(i, j) = C_2(i, j) \) then \( D(i, j) = 0 \); otherwise, \( D(i, j) = 1 \).

A fixed size of Julia stream cipher should be designed with respect to high key sensitivity and short time in keystream generation.

On one hand, if \( K \) is too big, the time computation complexity in keystream generation will increase. What is worst, a large \( K \) value results in a long keystream, making the whole keystream has less operation with plaintext. At the end, its output will be difficult to maintain pseudorandomness, decreasing the plain sensitivity in general. In the experiments, we randomly selected 631 pixels of Mandrill image, changing one bit value in the location of Julia set integer 32 years, which is a very long time and practically infeasible. When the size of Julia set \( K \) is 10, the stream cipher is 800 bits length, and the corresponding key space is nearly \( 2^{272} \).

### 4.3 Sensitivity analysis

A secure encryption system should be sensitive for input data. At the same time, an ideal image encryption should be sensitive with respect to the secret key such that a single bit change in the key should produce a completely different encrypted image [17]. In this paper, we test key sensitivity by changing the key values. The different pixels percentage between the two ciphertexts is an index of key sensitivity. For plaintext sensitivity, if one bit of plaintext changed, the corresponding ciphertext should also be much different. The common standards to test plain image sensitivity are the number of pixels change rate (NPCR) and unified average changing intensity (UACI), defined as (14) and (15).

### Table 3 Time consumption and PSNR of three different images of the size 256 × 256

| Lena    | Peppers | Elain   | Time, s | PSNR  | Time, s | PSNR  | Time, s | PSNR  |
|---------|---------|---------|---------|-------|---------|-------|---------|-------|
| 0.234/0.015 | 32.483  | 0.281/0.016 | 32.24 | 0.296/0.016 | 33.327 |

### Table 4 Seven secret keys of encryption

| Keys | Role                      | Type | Space, bits |
|------|----------------------------|------|-------------|
| \( m \) | Julia set for stream cipher | double  | 64          |
| \( p \) | Julia set for stream cipher | double  | 64          |
| \( q \) | Julia set for stream cipher | double  | 64          |
| \( X \) | location of Julia set     | integer | 32          |
| \( Y \) | location of Julia set     | integer | 32          |
| \( \text{key}_1 \) | diffusion                 | integer | 32          |
| \( \text{key}_2 \) | diffusion                 | integer | 32          |
set may not change, making the sensitivity for cipher to key weak. Table 5 demonstrates the experimental results.

Fig. 9 shows the results of key sensitivity with a certain key changed. In Fig. 9, the keys sensitivities of \( p \) and \( q \) both achieve \( 10^{-16} \) and that of \( m \) achieves \( 10^{-15} \). In addition, the wrong keys cannot decrypt the ciphertext correctly. As what we have anticipated, the key has a high sensitivity.

Fig. 8 Distributions of NPCRs and UACIs of Mandrill image with different \( K \)

\( a \) Distributions of NPCRs and UACIs with \( K = 8 \). The average NPCR is 99.62\% and the average UACI is 33.38\%.

\( b \) Distributions of NPCRs and UACIs with \( K = 10 \). The average NPCR is 99.52\% and the average UACI is 33.52\%.

\( c \) Distributions of NPCRs and UACIs with \( K = 32 \). The average NPCR is 98.27\% and the average UACI is 32.98\%.

| \( K \) | Different pixels percentage |
|---|---|
| 4 | 0 |
| 6 | 0 |
| 8 | 99.5514\% |
| 10 | 99.6033\% |

Table 5 Different pixels percentage between two ciphertexts with \( p \) changed from 0.5 to 0.5000000000000001
4.4 Cipher randomness analysis

Table 6 shows that our encrypted data of the 512 × 512 Lena image passes the sp800–22 test suite. It proves that the ciphertext has a good randomness.

| KEYS | different pixel percentage | Decrypted image |
|------|---------------------------|-----------------|
| P=0.5 | 0 | ![Decrypted image](image1) |
| Q=0.7 | 99.6023% | ![Decrypted image](image2) |
| M=15 | 99.6512% | ![Decrypted image](image3) |
| F0=13 | 99.5954% | ![Decrypted image](image4) |
| F100=29 | 99.5954% | ![Decrypted image](image5) |
| X=205 | 99.5954% | ![Decrypted image](image6) |
| Y=34 | 99.5954% | ![Decrypted image](image7) |

From Table 7, we can see that for Lena image of size 256 × 256, the encryption time is 31/234 ≃ 13.25% of the compression time. For the Boat image of size 512 × 512, the encryption time is 110/949 ≃ 11.59% of the compression time. They are no more than 15% [5]. Moreover, the time ratio is lower when the image size is bigger, that is, the encryption operation does not delay the compression process significantly. This property makes it easy to realise real-time compression and encryption.

5 Conclusions

In this paper, we have proposed a novel compression–encryption scheme based on the fractal dictionary and Julia set. In this scheme, the image compression based on fractal dictionary not only saves time consumption, but also gives a good quality of image reconstruction with a satisfying PSNR. Furthermore, as one iteration is required in decompression, the restored image can be accomplished in a flash. In the encryption process, the key has nearly $2^{272}$ key space with a 10 × 10 Julia set. As discussed in Section 4.3, the encryption system has a high plaintext sensitivity and key sensitivity to a minor perturbation. Subsequently, we carried out a detailed analysis of the Julia set size, and the experimental results show that when $K$ is equal to 8 or 10, it has a high sensitivity for cipher to both key and
plaintext. Additionally, the ciphertext passed the sp800-22 test suite, proving that the ciphertext has a good randomness. Finally, we tested the encryption/decryption time consumption of different images. The experimental results demonstrate that the encryption operation is < 15% of compression process, meaning that it does not delay the compression process and this makes it easy to realise real-time compression and encryption.

For future work, we will consider choosing better Julia set keys and other methods to improve the algorithm for the key conversion.

### Table 6: Ciphertext result of sp800-22 test suite

| Statistical test                                    | P-value | Result |
|-----------------------------------------------------|---------|--------|
| frequency                                           | 0.860274| SUCCESS |
| block frequency                                     | 0.560222| SUCCESS |
| runs                                                | 0.346199| SUCCESS |
| long runs of one’s (M= 781, N= 128)                 | 0.954460| SUCCESS |
| spectral discrete Fourier transform                  | 0.732205| SUCCESS |
| no overlapping templates (m = 9, B = 000000001)      | 0.646259| SUCCESS |
| overlapping templates (m = 9, M = 1032, N = 1016)   | 0.730268| SUCCESS |
| universal (L = 7, Q = 1280, K = 14 1577)            | 0.694044| SUCCESS |
| linear complexity (M = 500)                         | 0.166080| SUCCESS |
| serial (m = 16)                                     | 0.744639| SUCCESS |
| approximate entropy (m = 10)                        | 0.217446| SUCCESS |
| cumulative sums                                      | 0.581458| SUCCESS |
| random excursions (x = 1)                           | 0.566890| SUCCESS |
| random excursions variant (x = 1)                   | 0.972329| SUCCESS |
| rank                                                | 0.865548| SUCCESS |

### Table 7: Encryption/decryption time consumption

| Images                | Encryption time, ms | Decryption time, ms |
|-----------------------|---------------------|---------------------|
| Lena 256 × 256        | 31                  | 31                  |
| Peppers 256 × 256     | 31                  | 46                  |
| Elaine 512 × 512      | 125                 | 172                 |
| Boat 512 × 512        | 110                 | 125                 |
6 Acknowledgments

This research is supported by the National Natural Science Foundation of China (nos. 61103147, 61075018 and 61070098), the National Key Project of Science and Technology of China (no. 2011ZX05039–003-4) and the Fundamental Research Funds for the Central Universities (no. DUT12JB06).

7 References

1 Ahmed, F., Siyal, M.Y., Abbas, V.U.: ‘A perceptually scalable and jpeg compression tolerant image encryption scheme’. 2010 Fourth Pacific-Rim Symp. on Image and Video Technology (PSIVT), IEEE, 2010, pp. 232–238
2 Yuen, C.H., Wong, K.W.: ’A chaos-based joint image compression and encryption scheme using DCT and SHA-1’, Appl. Soft Comput., 2011, 11, (8), pp. 5092–5098
3 Hermassi, H., Rhouma, R., Belghith, S.: ’Joint compression and encryption using chaotically mutated Huffman trees’, Commun. Nonlinear Sci. Numer. Simul., 2010, 15, (10), pp. 2987–2999
4 Bourbakis, N.G.: ’Image data compression-encryption using G-scan patterns’. IEEE Int. Conf. on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation, IEEE, 1997, no. 2, pp. 1117–1120
5 Lian, S., Chen, X., Ye, D.: ’Secure fractal image coding based on fractal parameter encryption’, Fractals, 2009, 17, (02), pp. 149–160
6 Lock, A.J.J., Loh, C.H., Juhari, S.H., Samsudin, A.: ’Compression-encryption based on fractal geometric’. 2010 Second Int. Conf. on Computer Research and Development, IEEE, 2010, pp. 213–217
7 Barnsley, M.F., Demko, S.: ’Iterated function systems and the global construction of fractals’, Proc. R. Soc. Lond. A. Math. Phys. Sci., 1985, 399, (1817), pp. 243–275
8 Delp, E.J., Mitchell, O.R.: ’Image compression using block truncation coding’, IEEE Trans. Commun., 1979, 27, pp. 1335–1342
9 Wohlberg, B., De Jager, G.: ’A review of the fractal image coding literature’, IEEE Trans. Image Process. , 1999, 8, (12), pp. 1716–1729
10 Wang, J., Zheng, N.: ’A novel fractal image compression scheme with block classification and sorting based on Pearson’s correlation coefficient’, IEEE Trans. Image Process., 2013, 22, (9), pp. 3690–3792
11 Jafervazdeh, K., Kiani, K., Mozaffari, S.: ’Acceleration of fractal image compression using fuzzy clustering and discrete-cosine-transform-based metric’, IET Image Process., 2012, 6, (7), pp. 1024–1030
12 Sun, Y., Kong, R.: ’The research on the image encryption and coding algorithms based on M-J fractal sets’, Comput. Eng., 2013, 39, (1), pp. 230–233 (in Chinese)
13 Jacquin, A.E.: ’A novel fractal block-coding technique for digital images’. Int. Conf. on Acoustics, Speech, and Signal Processing, ICASSP-90, IEEE, 1990, pp. 2225–2228
14 Rapeti, S.A.: ’NLFS: a new non-linear feedback stream cipher’. Indian Institute of Technology, 2008
15 Chen, G., Mao, Y., Chui, C.K.: ’A symmetric image encryption scheme based on 3D chaotic cat maps’, Chaos Solitons Fractals, 2004, 21, (3), pp. 749–761
16 Available at http://www.sipi.usc.edu/database/database.php?volume=misc
17 Ahmad, J., Ahmed, F.: ’Efficiency analysis and security evaluation of image encryption schemes’, Computing, 2010, 23, p. 25