3D Res (2018) 9:3
https://doi.org/10.1007/s13319-017-0154-7

3DR REVIEW

Test and Verification of AES Used for Image Encryption

Yong Zhang

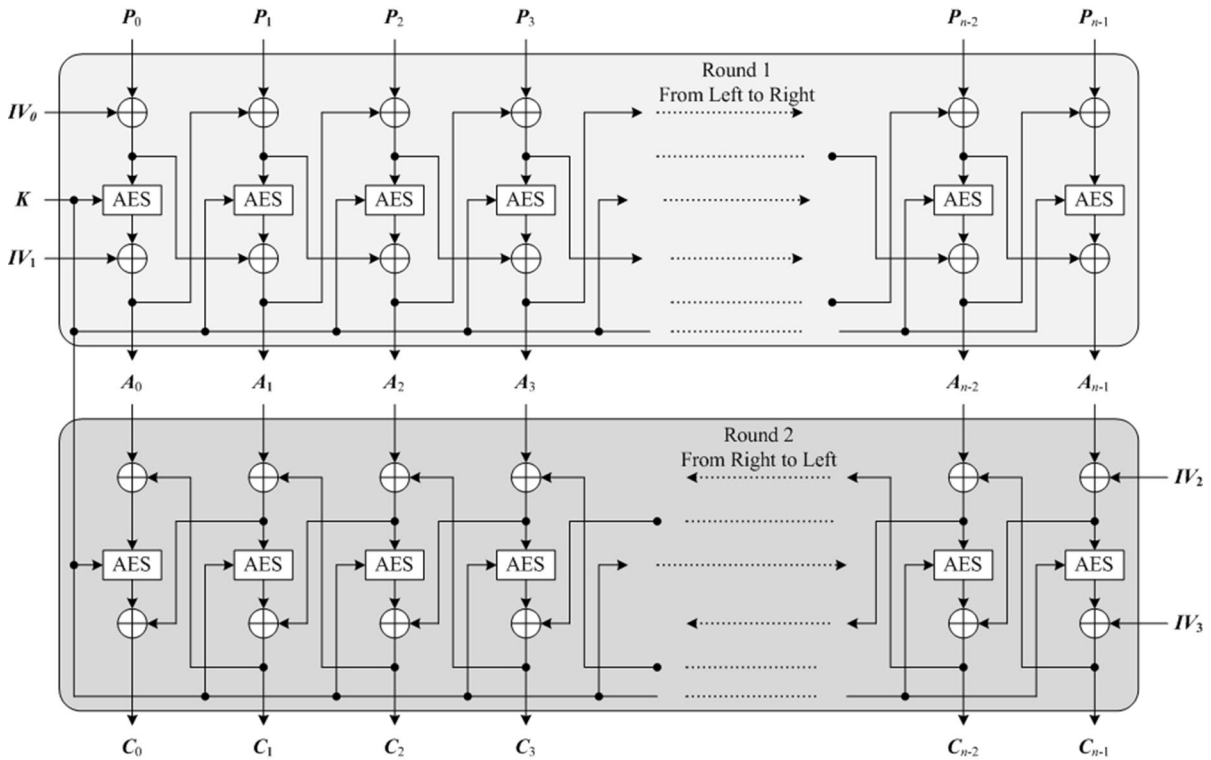
Received: 29 August 2017 / Revised: 26 December 2017 / Accepted: 28 December 2017

Abstract In this paper, an image encryption program based on AES in cipher block chaining mode was designed with C language. The encryption/decryption speed and security performance of AES based image cryptosystem were tested and used to compare the proposed cryptosystem with some existing image cryptosystems based on chaos. Simulation results show that AES can apply to image encryption, which refutes the widely accepted point of view that AES is not suitable for image encryption. This paper also suggests taking the speed of AES based image encryption as the speed benchmark of image encryption algorithms. And those image encryption algorithms whose speeds are lower than the benchmark should be discarded in practical communications.

Y. Zhang (✉)
School of Software and Communication Engineering, Jiangxi University of Finance and Economics, Nanchang, People's Republic of China
e-mail: zhangyong@jxufe.edu.cn

Published online: 12 January 2018

Graphical Abstract Image cryptosystem based on AES in CBC mode



Keywords Information security · Image encryption · AES—advanced encryption standard · CBC—cipher block chaining

1 Introduction

In 2002, the advance encryption standard (AES) became the encryption standard of U.S. government, and replaced the data encryption standard (DES) proposed in 1977 [1]. AES, also known as Rijndael algorithm, is a fast symmetric block cipher oriented to optimized software implementation [2]. For AES, its secret key can be 128, 192 or 256 bits long, and its plaintext block and cipher block are 128 bits long [3, 4].

Researches based on chaotic systems are popular in the image encryption field [5–11], because encrypting digital images needs a large number of key streams while chaotic systems can generate multiple pseudo-random numbers easily. Meanwhile, some image cryptosystems based on chaos have exposed variety of shortcomings in fighting against the chosen/known plaintext attacks [12–15].

Recently, Hua and Zhou proposed a new two-dimensional Logistic-modulated-Sine map, and then used it to generate the chaotic matrices to perform image confusion and diffusion operations in the bit level [16]. Eslami et al. presented an improvement over the permutation–diffusion architecture of image cryptosystem based on total shuffling [17]. The improved scheme can resist differential attack by employing the intermediate pixels to diffuse the resultant pixels. Cheng et al. used the chaotic tent map, lookup table and S-box of AES to generate a pseudo-random sequence for image encryption, which avoided the operations of floating-point numbers [18]. In [19], a new three-dimensional chaotic system was proposed to develop the S-box for image encryption. And, in [20], a novel confusion method based on both irregular wave-line and SHA-3 was suggested to encrypt the plain images. This scheme can resist the

chosen-plaintext attack but with slow encryption speed.

In the above researches, the typical time domain algorithms, such as XOR or add-modulus operations in finite domain, are used to transform the plain images into noise-like images with the help of chaotic pseudo-random numbers. Many scholars believed that AES was unsuitable for image data encryption [21–25], because that images are typical of huge volume, strong redundancy and high correlation among adjacent pixels. To some extent, they thought that AES based image encryption algorithm was slow in speed on general computers.

Investigation shows that the statement—AES is safe on image encryption but slow in general computer—is well-accepted [26–32]. For example, Fawaz et al. pointed out that the traditional techniques, such as DES and AES, are not efficient on the encryption of digital images [30]. This may mislead the new entrants to the image encryption field, making them underestimate the power of AES. Therefore, publishing the detailed research result of AES on image encryption is necessary.

According to Shannon, image encryption requires both confusion and diffusion operations [33]. However, AES in cipher block chaining (CBC) mode only need to perform the diffusion operation on the whole image, while omit the overall image permutation (confusion is realized in each block of image), which maybe another reason that make AES is considered to be non-suitable for image encryption. In 2008, Li et al. pointed out that the permutation-only image cryptosystems are weak against the chosen-plaintext attacks [34], and in 2016, Jolfaei et al. [35] suggested a quantitative cryptanalysis method on the basis of [34]. So the diffusion operation is necessary in the image cryptosystem. Through this paper, we will prove that AES in CBC mode can achieve optimal diffusion to reach the security standard of image cryptosystem.

This paper studied the image encryption scheme based on AES in CBC mode, and then compared AES based scheme with some chaos based image encryption schemes. The results show that AES based image encryption is secure and fast in general computer. So, we propose to use the speed of AES based image encryption as the lower speed limit standard for image encryption schemes based on chaos. Those schemes slower than AES should be discarded in practical communications. The paper is organized as follows: Sect. 2 introduces two image schemes based on AES, i.e. AES-S and AES-D; Sect. 3 gives the simulation results of image encryption with AES-S and AES-D; Sect. 4 compares the speeds of both AES-S and AES-D with those of schemes based on chaos, and describes a fair speed comparison method for different computers and programmers; Sect. 5 analyzes the security performance of AES based schemes; Sect. 6 summarizes the full paper.

2 AES Based Image Cryptosystem

Currently, AES is the most widely used symmetric cipher, which transforms a 128 bits plain block into a cipher block with the same length. The length of the secret key can be 128, 192 or 256 bits, and the corresponding AES is called as AES-128, AES-192 or AES-256. The longer secret key of AES is corresponding to the more rounds and the slower encryption speed. So AES-256 is the slowest, and AES-192 is slower than AES-128. It is worth mentioning that there is no official report on security issue of AES.

The structure of AES is as shown in Fig. 1, which includes N_r rounds of round functions and one key expansion. For AES-128, AES-192 and AES-256, N_r is set to 10, 12 and 14, respectively. The standard round function is composed of four different byte-oriented transformations, namely, SubBytes, ShiftRows, MixColumns and XOR-operation (i.e. $\oplus$ in Fig. 1). In Fig. 1b, ShiftRows^{-1} means the inverse of ShiftRows. The reader is suggested to refer to Refs. [1, 2] for detailed description of AES.

The image cryptosystem based on AES in CBC mode is as shown in Fig. 2. Where, the secret key $\boldsymbol{K}$ of AES is the key of the whole system, and $\boldsymbol{IV}_i$, $i = 0, 1, 2, 3$, are four arbitrarily public initial vectors, all of length 128 bits. For the sake of brevity, we set all the $\boldsymbol{IV}$s to zero vectors.

In Fig. 2a, the image encryption part only containing "Round 1" is called as AES-S, and the whole system containing "Round 1" and "Round 2" is called as AES-D. In Fig. 2b, AES_d represents the inverse of AES. The encryption steps of AES-S and AES-D will be described in the following. We assume that the input image is an 8-bit grayscale one denoted by

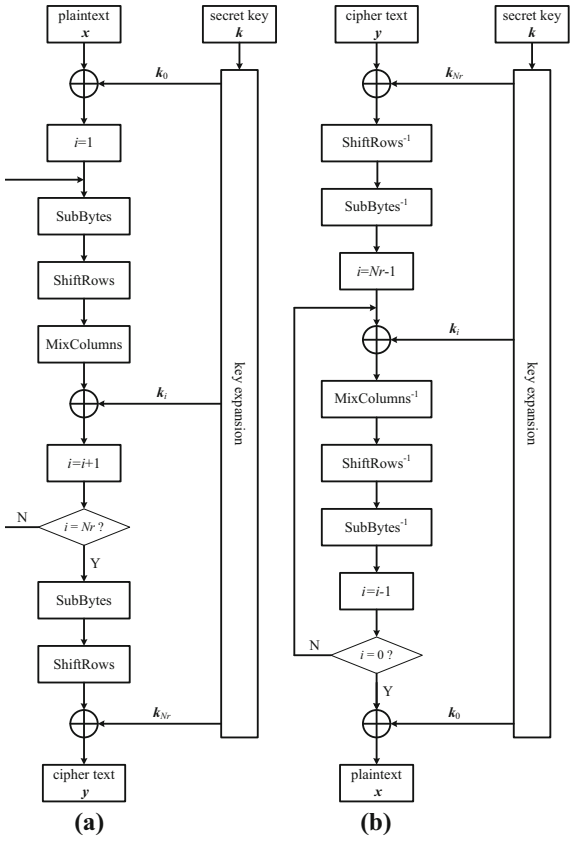


Fig. 1 AES and its inverse. **a** AES (i.e. encryption process), **b** inverse of AES (i.e. decryption process)

$\boldsymbol{P}$ with the size of $M \times N$. Where, M and N are the height and width of the image, respectively.

The encryption process of AES-S is as follows:

Step 1 Expands $\boldsymbol{P}$ row by row into a vector, denoted by $\boldsymbol{V}(0 \text{ to } MN-1)$.

Denote $\text{mod}(MN,16) = r$. Then pad $\boldsymbol{V}$ with $32-r$ bytes, such that $\boldsymbol{V}(MN) = 255$, $\boldsymbol{V}(i) = 0$, $i = MN+1, \quad MN+2,\ldots,MN+15-r$. And $\boldsymbol{V}(MN+16-r \text{ to } MN+31-r)$ of length 128 bits saves the value of N. So, the new $\boldsymbol{V}$ is of length $n = MN+32-r$. Divide $\boldsymbol{V}$ into blocks of each 16 bytes, sequentially denoted by $\boldsymbol{P}_i$, $i = 0,1,2,\ldots,n-1$. These blocks are inputs of encryption system, as shown in Fig. 2.

Step 2. Encrypt $\boldsymbol{P}_i$, $i = 0, 1, 2, \ldots, n-1$, with the following formulae

$$\boldsymbol{A}_0 = \boldsymbol{IV}_1 \text{XOR AES}\,(\boldsymbol{K}, \boldsymbol{P}_0 \text{XOR}\, \boldsymbol{IV}_0) \tag{1}$$

$$\boldsymbol{A}_1 = (\boldsymbol{IV}_0 \text{XOR} \boldsymbol{P}_0)\ \text{XOR AES}(\boldsymbol{K}, \boldsymbol{P}_1 \text{XOR}\, \boldsymbol{A}_0) \tag{2}$$

$$\boldsymbol{A}_i = (\boldsymbol{A}_{i-2} \text{XOR} \boldsymbol{P}_{i-1})\ \text{XOR AES}(\boldsymbol{K}, \boldsymbol{P}_i \text{XOR} \boldsymbol{A}_{i-1}),\quad i = 2,3,\ldots,n-1 \tag{3}$$

where AES($\boldsymbol{K}$,$\boldsymbol{X}$) represents encrypting the block $\boldsymbol{X}$ with the key $\boldsymbol{K}$, and XOR means bitwise exclusive-or.

The resultant $\boldsymbol{A}_i$, $i = 0, 1, 2, \ldots, n-1$, are the cipher image of AES-S.

As can be seen from Fig. 2, AES-S is included in AES-D system, and its output is the input of "Round 2" of AES-D. The encryption process of AES-D is as follows:

Step 1 Execute AES-S system to encrypt plain image $\boldsymbol{P}$ into the intermediate cipher blocks, denoted by $\boldsymbol{A}_i$, $i = 0, 1, 2, \ldots, n-1$. This step is the process of AES-S.

Step 2 Encrypt $\boldsymbol{A}_i$, $i = n-1, n-2, \ldots, 2, 1, 0$, sequentially with the following formulae

$$\boldsymbol{C}_{n-1} = \boldsymbol{IV}_3 \text{XOR AES}\,(\boldsymbol{K}, \boldsymbol{A}_{n-1} \text{XOR}\, \boldsymbol{IV}_2) \tag{4}$$

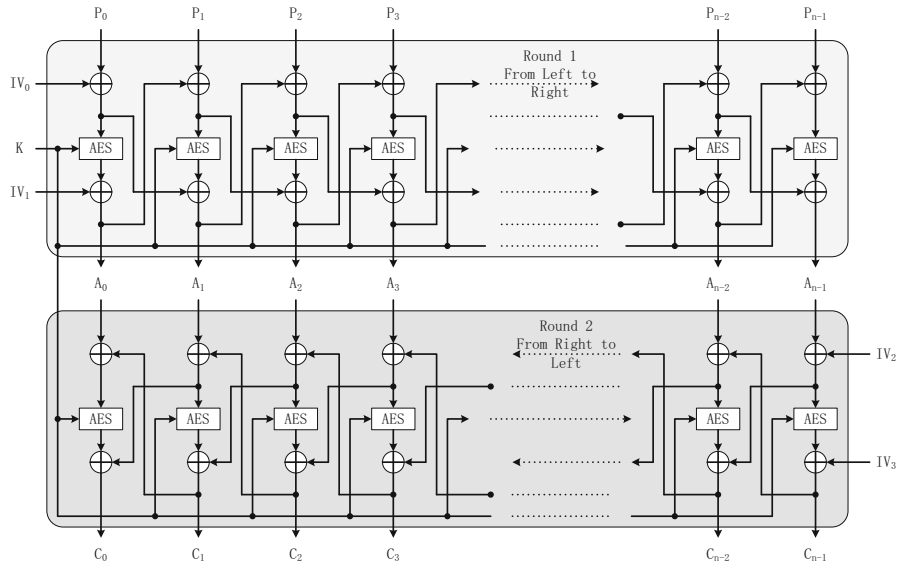
$$\boldsymbol{C}_{n-2} = (\boldsymbol{IV}_2 \text{XOR}\, \boldsymbol{A}_{n-1})\ \text{XOR AES}(\boldsymbol{K}, \boldsymbol{A}_{n-2} \text{XOR}\, \boldsymbol{C}_{n-1}) \tag{5}$$

$$\boldsymbol{C}_i = (\boldsymbol{C}_{i+2} \text{XOR} \boldsymbol{A}_{i+1})\ \text{XOR AES}(\boldsymbol{K}, \boldsymbol{A}_i \text{XOR}\, \boldsymbol{C}_{i+1}),\quad i = n-3, n-4, \ldots, 1, 0 \tag{6}$$

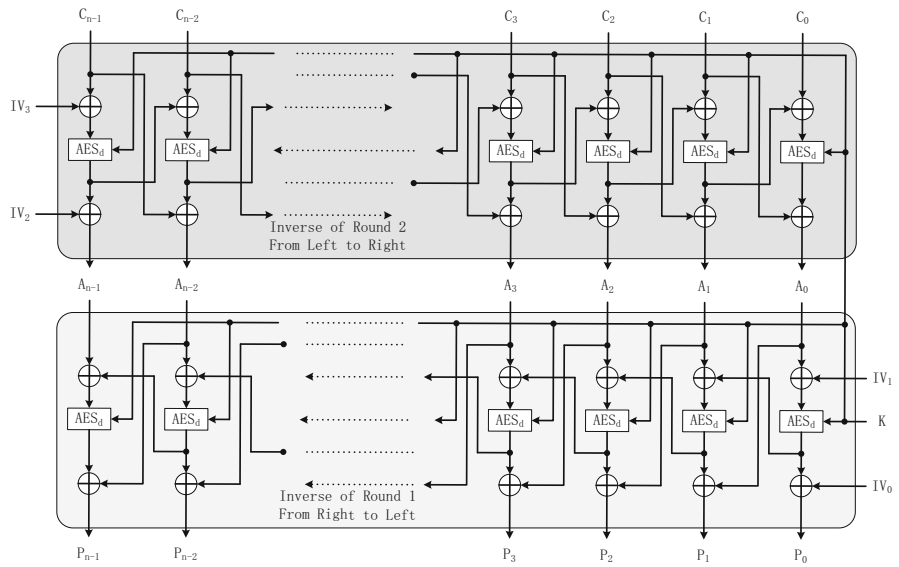
Here, $\boldsymbol{C}_i$,$i = 0, 1, \ldots, n-1$, are the cipher image of AES-D system.

Remark 1 AES algorithm is secure enough to resist all kinds of passive attacks. From Fig. 2, we can see that the diffusion of AES-S is unidirectional, and the information of $\boldsymbol{P}_i$ cannot be diffused into $\boldsymbol{A}_j$ $(j < i)$. So, the encryption strength of AES-S is equivalent to that of single AES.

Remark 2 AES-D system can be considered as the combination of two AES-S systems, and realizes bidirectional information diffusion. The information of each plain block can be diffused into the entire cipher image. So, the encryption intensity of AES-D is equivalent to that of a cascade of $2n$ AES algorithms.



(a)



(b)

Fig. 2 Image cryptosystem based on AES in CBC mode. **a** Encryption process, **b** decryption process

The image decryption system based on AES is the reverse of encryption system as shown in Fig. 2. For the reverse system of AES-S, the decryption algorithm is as described as follows.

$$\boldsymbol{R}_0 = \boldsymbol{IV}_0 \mathrm{XOR}\ \mathrm{AES_d}(\boldsymbol{K}, \boldsymbol{A}_0 \mathrm{XOR}\ \boldsymbol{IV}_1) \tag{7}$$

$$\boldsymbol{R}_1 = \boldsymbol{A}_0 \mathrm{XOR}\ \mathrm{AES_d}(\boldsymbol{K}, \boldsymbol{A}_1 \mathrm{XOR}\ (\boldsymbol{P}_0\ \mathrm{XOR}\ \boldsymbol{IV}_0)) \tag{8}$$

$$\boldsymbol{R}_i = \boldsymbol{A}_{i-1} \mathrm{XOR}\ \mathrm{AES_d}(\boldsymbol{K}, \boldsymbol{A}_i \mathrm{XOR}\ (\boldsymbol{P}_{i-1} \mathrm{XOR}\ \boldsymbol{A}_{i-2})), \quad i = 2, 3, \ldots, n-1 \tag{9}$$

where $AES_d(\boldsymbol{K},\boldsymbol{Y})$ means using the decryption algorithm of AES to decrypt $\boldsymbol{Y}$ with the key $\boldsymbol{K}$.

For the reverse system of AES-D, the decryption process starts with the following Eqs. (10–12) to obtain $\boldsymbol{A}_i$, $i = n-1, n-2, \ldots, 1, 0$, and then executes the reverse system of AES-S to get $\boldsymbol{R}_i, i = 0,1, \ldots, n-1$, with Eqs. (7–9).

$$\boldsymbol{A}_{n-1} = \boldsymbol{IV}_2 \mathrm{XOR}\ AES_d(\boldsymbol{K}, \boldsymbol{C}_{n-1} \mathrm{XOR}\ \boldsymbol{IV}_3) \tag{10}$$

$$\boldsymbol{A}_{n-2} = \boldsymbol{C}_{n-1} \mathrm{XOR}\ AES_d(\boldsymbol{K}, \boldsymbol{C}_{n-2} \mathrm{XOR}\ (\boldsymbol{A}_{n-1} \mathrm{XOR}\ \boldsymbol{IV}_2)) \tag{11}$$

$$\boldsymbol{A}_i = \boldsymbol{C}_{i+1} \mathrm{XOR}\ AES_d(\boldsymbol{K}, \boldsymbol{C}_i \mathrm{XOR}\ (\boldsymbol{A}_{i+1} \mathrm{XOR}\ \boldsymbol{C}_{i+2})), \quad i = n-3, n-4, \ldots, 2, 1, 0 \tag{12}$$

$\boldsymbol{R}_{n-16} \sim \boldsymbol{R}_{n-1}$ (of total 128 bits) preserve the value of N. The location of the first emergence of 255 in $\boldsymbol{R}_{n-17} \sim \boldsymbol{R}_{n-32}$ is denoted by k, and then $M = k/N$. If $N > 16$, then we can also use $M = \mathrm{floor}((n-16)/N)$ to calculate M, where, floor(x) rounds x towards minus infinity. Then, convert the sequence $\boldsymbol{R}_0 \sim \boldsymbol{R}_{MN-1}$ into a matrix $\boldsymbol{R}$, such that $\boldsymbol{R}(i,j) = \boldsymbol{R}_{iN+j}$, $i = 0,1,\ldots,M-1$, $j = 0,1,\ldots,N-1$. The matrix $\boldsymbol{R}$ is the recovered image.

3 Simulation Test

The computer used is configured with Intel(R) Core(TM) i7-4720HQ CPU@2.60 GHz, 8 GB DDR3L Memory, Windows 10 (64-bit), Eclipse C/C++ with MinGW. To save space, we only give the image encryption/decryption results of AES-S and AES-D with AES-256. The plain images used are 8-bit grayscale Lena, Pepper, all-black and all-white ones all of size 256 × 256 as shown in Fig. 3. Without loss of generality, the secret key of length 256 bits is taken as "{35, 65, 101, 206, 130, 192, 147, 138, 79, 122, 67, 136, 65, 112, 81, 151, 101, 18, 61, 170, 211, 203, 125, 70, 78, 204, 178, 250, 37, 120, 181, 24}" (in decimal format). The simulation results of AES-S and AES-D are as shown in Figs. 4 and 5, respectively. For the plain image of size 256 × 256, the corresponding cipher image is of size 256 × 256 + 32. To display the cipher image in rectangular format, we add 224 bytes in the rear of each cipher image (the first byte is 255, and the remaining 223 bytes are all 0). So, each cipher image in Figs. 4a–d and 5a–d has a black line in its last row. The histograms of plain and cipher images (as shown in Figs. 3a–d, 4a–d, 5a–d, respectively) are as shown in Fig. 6a–l, respectively.

As can be seen from Figs. 3, 4, AES-S can encrypt the plain images (as shown in Fig. 3a–d, respectively) into noise-like cipher images (as shown in Fig. 4a–d, respectively), and similarly, the reverse system of AES-S can decrypt the cipher images (as shown in Fig. 4a–d, respectively) into the original plain images (as shown in Fig. 4e–h, respectively). As can be seen from Figs. 3 and 5, AES-D can encrypt the plain images (as shown in Fig. 3a–d, respectively) into noise-like cipher images (as shown in Fig. 5a–d, respectively), and similarly, the reverse system of AES-D can decrypt the cipher images (as shown in Fig. 5a–d, respectively) into the original plain images (as shown in Fig. 5e–h, respectively). These demonstrate that both AES-S and AES-D systems work properly.

Figure 6 shows that the histograms of plain images have obvious fluctuation (as shown in Fig. 6a–d), while the histograms of cipher images are fairly flat (as shown in Fig. 6e–l). The cipher images hide the statistical characteristics of each pixel, which makes the attack based on histogram impossible.

4 Encryption/Decryption Speed and Speed Comparison Analysis

Encryption and decryption speed is an important indicator of image cryptosystem. As we all know, it is very easy to design an image encryption system with extreme security when the speed indicator can be ignored. But low speed will make the image cryptosystem completely lose the application value in practical communications. However, there is no existing standard to measure the speed of image encryption. Since each image cryptosystems are implemented on different computers with different configuration, and even on same computer, the capacities of the programmers to optimize the program are different, so it is hard to fairly compare the speeds of different image cryptosystems.

AES is the encryption standard of the U.S. government, and in fact, is also a global standard for symmetric cipher. We suggest using the speed of AES based image cryptosystem as the lower speed

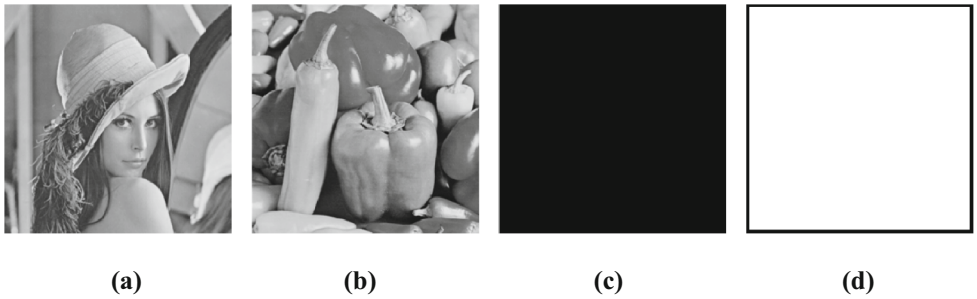


Fig. 3 Plain images. **a** Lena, **b** pepper, **c** all-black and **d** all-white

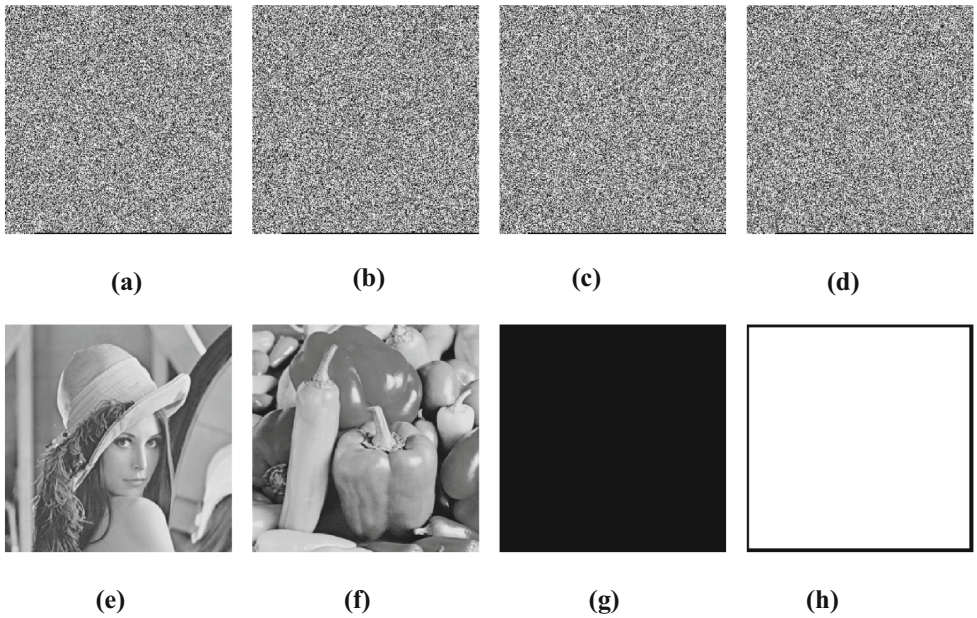


Fig. 4 Test results with AES-S system. **a**–**d** Encrypted images of Fig. 3a–d, respectively, **e**–**h** recovered images of **a**–**d**, respectively

limit of image encryption. Considering that the computers used by scholars may be different and the capacities of programmers are different, we think it unreasonable to directly use the speed of AES based image cryptosystem as the speed standard. So, we employ the standard FFT algorithm, named by FFTW [36], as the encryption speed standard. FFTW is a fast C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (see http://www.fftw.org). MATLAB 9 calls the library of FFTW to perform fast Fourier transform (known as function *fft*), so we directly use the function *fft* in MATLAB 9 as FFTW. In MATLAB 9, we use the 100 Hz cosine signal 'cos(200πt)' to generate a sequence of length 2^{20} with the sampling rate 1000 Hz, and then do fast Fourier transform with function *fft* on the sequence. The consumed time by *fft* is denoted by τ_0.

Obviously, the values of τ_0 are different for different computers. For example, in our computer described in Sect. 3, $\tau_0 = 0.0186$ s. Now, in the same computer, we encrypt/decrypt an 8-bit grayscale image of size 256×256 with AES-S, and then denote the time consumed as τ_{es} and τ_{ds}, respectively. So, we can get $TH_{es} = \tau_{es}/\tau_0$ and $TH_{ds} = \tau_{ds}/\tau_0$, named by the first thresholds for encryption and decryption, respectively. Also, in the same computer, we encrypt/decrypt an 8-bit grayscale image of size 256×256 with AES-D, and then denote the time consumed as τ_{ed}

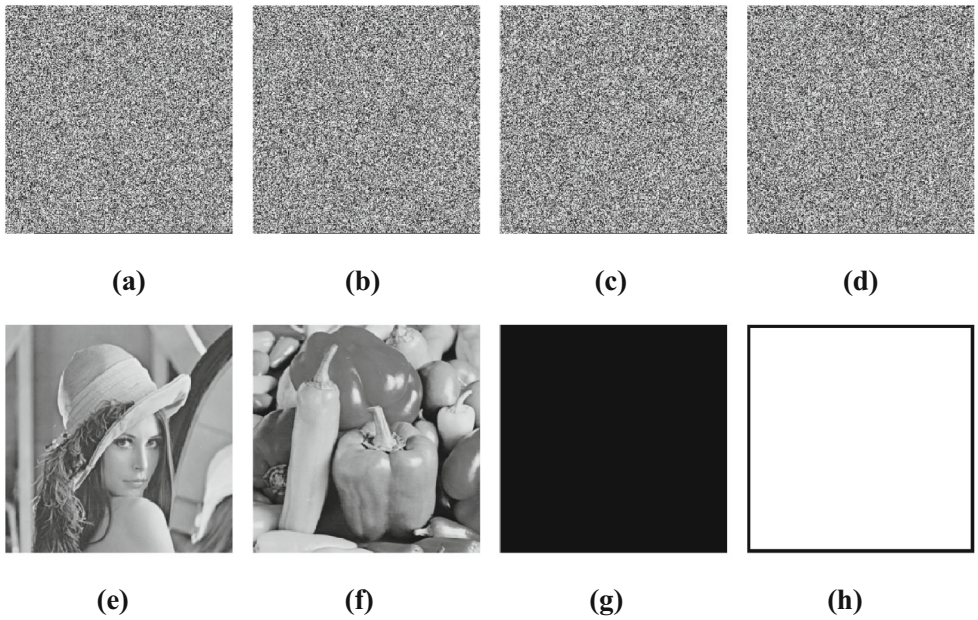


Fig. 5 Test results with AES-D system. **a–d** Encrypted images of Fig. 3a–d, respectively, **e–h** recovered images of **a–d**, respectively

and τ_{dd}, respectively. So, we can get $TH_{ed} = \tau_{ed}/\tau_0$ and $TH_{dd} = \tau_{dd}/\tau_0$, named by the second thresholds for encryption and decryption, respectively. As for a new proposed image cryptosystem, the τ_0 and its encryption time τ_e and decryption time τ_d for 8-bit grayscale image of size 256×256 can be obtained on any computer, and then $TH_e = \tau_e/\tau_0$ and $TH_d = \tau_d/\tau_0$ can be obtained. If $TH_e < TH_{es}$ and $TH_d < TH_{ds}$, the speed of the new cryptosystem is excellent; else if $TH_e < TH_{ed}$ and $TH_d < TH_{dd}$, the speed of new cryptosystem is desirable; else, the speed of new cryptosystem is unsatisfied.

On our computer, we implemented AES-S and AES-D with C language (see "Appendix"), and calculated the first and second thresholds, as shown in Table 1. Five of the recently proposed image cryptosystems [16–20] also were studied in this paper, the values of their TH_e and TH_d are included in Table 1 too. The reason for they are selected is that they were proposed recently and were widely quoted. In Table 1, the values of TH_e and TH_d corresponding to AES-S with 256-bit key are taken as the first thresholds, and the values of TH_e and TH_d corresponding to AES-D with 256-bit key are taken as the second thresholds. The thresholds are shown in bold style in Table 1. So, $TH_{es} = 0.8065$, $TH_{ds} = 0.9140$, $TH_{ed} = 1.6129$ and $TH_{dd} = 1.8817$. The tested image is an 8-bit grayscale one of size 256×256.

The results in Table 1 may surprise many scholars. Perhaps there're better optimization methods, but we did our best to optimize the schemes in [16–20] with C language, we still get the results showing that the speed of AES based image cryptosystem is significantly superior to those schemes in [16–20]. The encryption speed in [19] is very fast, but its decryption speed is fairly slow. Except [19], all of other schemes' speed indicators are larger than the second thresholds and much larger than the first thresholds in Table 1. These indicate that these schemes are much slower than AES based image cryptosystem. Therefore, the speeds of those schemes should be improved to get the valuable in practical communications.

5 Security Performance

5.1 Security Analysis Based on AES

Although we are unable to prove in theoretically that AES is secure, there is no official report that AES is not secure up to now. So, it is reasonable to believe that AES can resist the brute-force attack and all kinds of passive attacks. Since AES is secure, AES-S is secure.

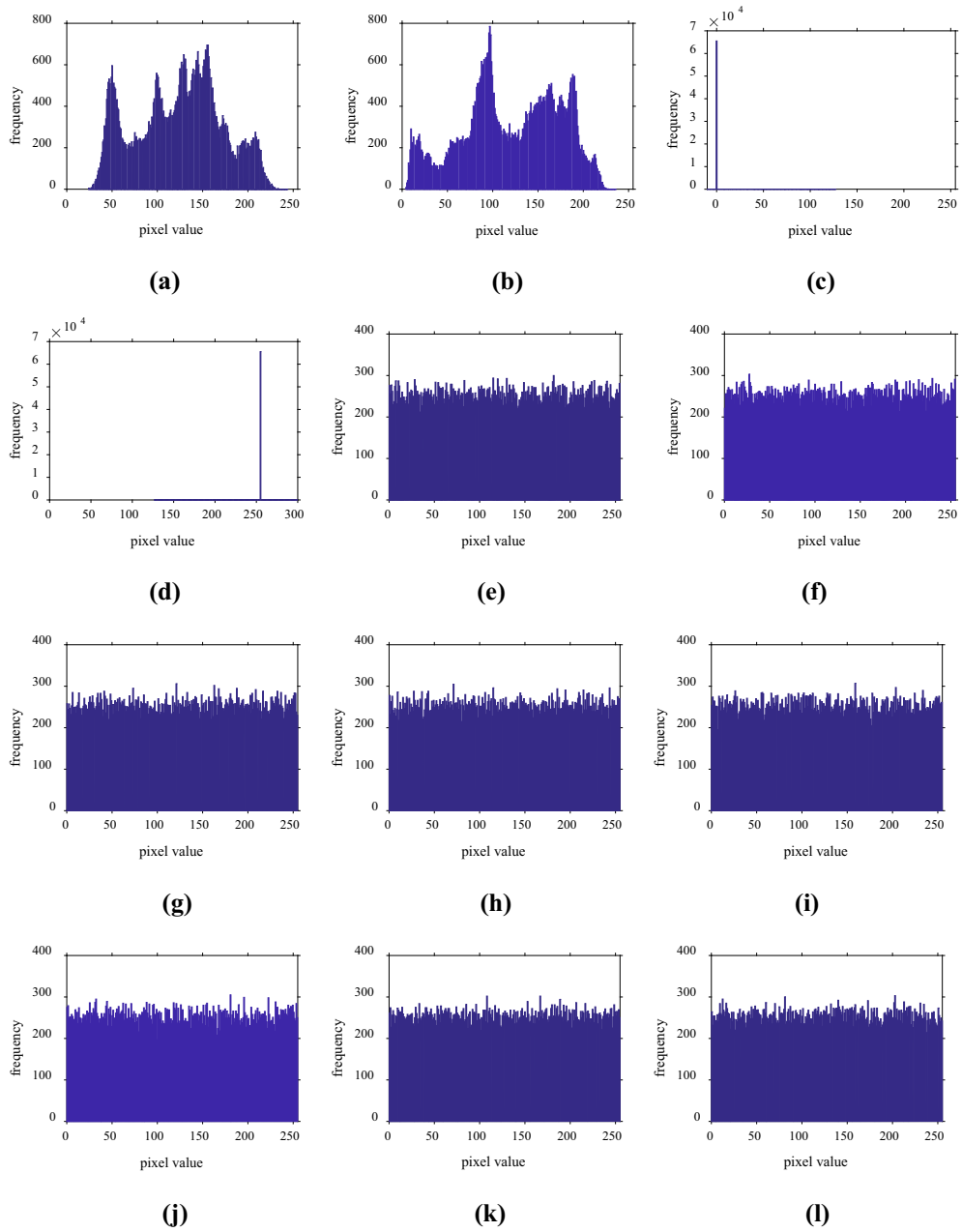


Fig. 6 Histograms test. **a–d** Histograms of Fig. 3a–d, respectively, **e–h** histograms of Fig. 4a–d, respectively, **i–l** histograms of Fig. 5a–d, respectively

AES-S system is typical AES in CBC mode with two public vectors $\boldsymbol{IV}_0$ and $\boldsymbol{IV}_1$. AES-S is equivalent to single AES on the secure intensity. While AES-D system includes two AES-S systems. Due to the round structure, AES-D is equivalent to a cascade of $2n$-AES. AES-D is securer than AES-S. In AES-D system, $\boldsymbol{IV}_0$, $\boldsymbol{IV}_1$, $\boldsymbol{IV}_2$ and $\boldsymbol{IV}_3$ are public vectors, and they, together with cipher image, are transferred to the receiver through public information channel.

5.2 Key Space Analysis

The key of AES-S or AES-D is the key of AES, which is of length 128, 192 or 256 bits. So, the size of key

Table 1 Encryption/decryption speed indicators comparison

Scheme	τ_e (s)	τ_d (s)	τ_0 (s)	TH_e	TH_d
AES-S					
128-bit key	0.011	0.012	0.0186	0.5914	0.6452
192-bit key	0.013	0.014	0.0186	0.6989	0.7527
256-bit key	0.015	0.017	0.0186	**0.8065**	**0.9140**
AES-D					
128-bit key	0.022	0.024	0.0186	1.1828	1.2903
192-bit key	0.026	0.028	0.0186	1.3978	1.5054
256-bit key	0.030	0.035	0.0186	**1.6129**	**1.8817**
Ref. [16]	0.080	0.080	0.0186	4.3011	4.3011
Ref. [17]	0.041	0.041	0.0186	2.2043	2.2043
Ref. [18]	0.044	0.044	0.0186	2.3656	2.3656
Ref. [19]	0.010	0.052	0.0186	0.5376	2.7957
Ref. [20]	2.313	2.313	0.0186	12.4355	12.4355

In eclipse C/C++ environment, the minimum time resolution unit of C executable program is 0.001 s. While, τ_0 is calculated by MATLAB 9, which precision is set to 0.0001 s

space of AES-S or AES-D is 2^{128}, 2^{192} or 2^{256}. According to the encryption speed in Table 1, for 128-bit key and 256 × 256 sized image, to crack AES-S and AES-D with the brute-force attack (i.e. to try half of the key space) on our computer will cost about 1.1869×10^{29} years and 2.3739×10^{29} years, respectively. Even if the future computer is 10^9 times faster than ours, the consumed time is at least 1.1869×10^{20} years, which is far greater than current human's lifespan. So, the key of size 128 bits is sufficient for image cryptosystem to against the exhaustive attack. And the image cryptosystem with the secret key of size 192 or 256 bits are more secure.

5.3 Statistical Characteristics Analysis

Statistical characteristics analysis includes histogram analysis, correlation analysis and information entropy. As can be seen in Fig. 6, the histograms of the cipher images produced by AES-S and AES-D are flat, indicating that the occurrence frequency of each gray value is approximately equal. Thus, the cipher images can frustrate the analysis based on histogram. In the following, we will analyze the correlation characteristics and information entropy.

5.3.1 Correlation Analysis

A good image encryption algorithm will destroy the correlation between adjacent pixels to eliminate the redundant information in plain image. In general, the larger the correlation coefficient is, the stronger the correlation of adjacent pixels is, and the greater the redundancy of information is. Due to the huge amount property of image data, it usually selects the finite pairs of adjacent pixels to calculate the correlation coefficient. For example, select K pairs of adjacent pixels from the image, and denote their values by (x_i,y_i), $i = 1, 2, \ldots, K$, and then calculate their correlation coefficient r_{xy} by the following formula.

$$r_{xy} = \frac{\mathrm{cov}(\boldsymbol{x},\boldsymbol{y})}{\sqrt{D(\boldsymbol{x})D(\boldsymbol{y})}} = \frac{\sum_{i=1}^{K}(x_i - E(\boldsymbol{x}))(y_i - E(\boldsymbol{y}))}{\sqrt{\left(\sum_{i=1}^{K}(x_i - E(\boldsymbol{x}))^2\right)\left(\sum_{i=1}^{K}(y_i - E(\boldsymbol{y}))^2\right)}} \quad (13)$$

where $\boldsymbol{x} = \{x_i\}, \boldsymbol{y} = \{y_i\}, i = 1, 2, \ldots, K$, $E(\boldsymbol{x})$ represents the mean of vector $\boldsymbol{x}$, $D(\boldsymbol{x})$ represents the variance of vector $\boldsymbol{x}$, and $\mathrm{cov}(\boldsymbol{x},\boldsymbol{y})$ returns the covariance between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

Without loss of generality, take the plain images (see Fig. 3a–d), the cipher images (see Fig. 4a–d) produced by AES-S and the cipher images (see Fig. 5a–d) produced by AES-D as examples. And let $K = 2000$. Select K pairs of adjacent pixels in the horizontal, vertical, diagonal and counter-diagonal directions, and calculate the correlation coefficients according to Eq. (13). Then, list the results in Table 2,

and illustrate the correlations of Figs. 3a, 4a, 5a in horizontal direction in Fig. 7.

It can be seen from Table 2 that the correlation coefficient of plain image Lena in each direction is close to 1, and the correlation coefficients of all-black and all-white images in all directions are 1. These show that the plain images have strong correlations in all directions. While the correlation coefficients of cipher images are all close to 0, indicating that the cipher images have eliminated the correlation between adjacent pixels.

Figure 7 is another powerful evidence of correlation. In Fig. 7a, the adjacent pixels of Lena in horizontal direction are distributed near the straight line $y = x$, and are in strong linear correlation. While in Fig. 7b–c, the adjacent pixels of cipher images in horizontal direction are distributed in the whole phase space, and are in no correlation.

5.3.2 *Information Entropy*

Histogram of image vividly depicts the distribution of gray values of pixels in the image, while information entropy quantifies the histogram in mathematical meanings. We assume that there are L grayscale levels in the image, and the occurrence frequency of each grayscale level is p_i, then information entropy H of image can be calculated by the following

$$H = -\sum_{i=0}^{L-1} p_i \log_2 p_i \tag{14}$$

For a true random image of $L = 256$ grayscale levels, its information entropy is $\log_2 L = 8$ bits. And the maximum value of information entropies of 256 grayscale images is 8 bits.

We calculated the information entropies of images in Fig. 6, and listed the results in Table 3.

In Table 3, the information entropies of plain images Lena and Pepper (whose histograms are as shown in Fig. 6a, b) both have a certain difference from theoretical value (i.e. 8), and the information entropies of all-black and all-white images are both 0 s. Whereas, the information entropies of cipher images (whose histograms are as shown in Fig. 6e–l, respectively) are very close to 8. Thus, Table 3 shows in mathematical that the cipher images have flat histograms, and can defeat the statistical analysis based on grayscale value distribution.

5.4 Sensitivity Analysis

The sensitivity analysis of image encryption system generally includes key sensitivity analysis and plaintext sensitivity analysis. For an encryption system, encrypt the same plain image with arbitrarily two slightly different keys, if the two produced cipher

Table 2 Results of correlation coefficients

Image	Horizontal	Vertical	Diagonal	Counter-diagonal
Lena				
Fig. 3a	0.971045	0.943174	0.922065	0.949810
Fig. 4a	0.046749	− 0.017328	− 0.007833	− 0.000899
Fig. 5a	0.030768	0.019044	0.010293	− 0.004687
Pepper				
Fig. 3b	0.959614	0.965931	0.927128	0.939008
Fig. 4b	0.005233	− 0.002849	− 0.014339	0.021487
Fig. 5b	− 0.043959	0.004757	0.003007	− 0.003926
All-black				
Fig. 3c	1.000000	1.000000	1.000000	1.000000
Fig. 4c	0.000285	0.027921	− 0.002654	− 0.005465
Fig. 5c	0.016699	− 0.010638	− 0.016424	0.015041
All-white				
Fig. 3d	1.000000	1.000000	1.000000	1.000000
Fig. 4d	0.006848	− 0.005482	− 0.023528	0.003298
Fig. 5d	− 0.011742	− 0.056329	0.000954	0.015618

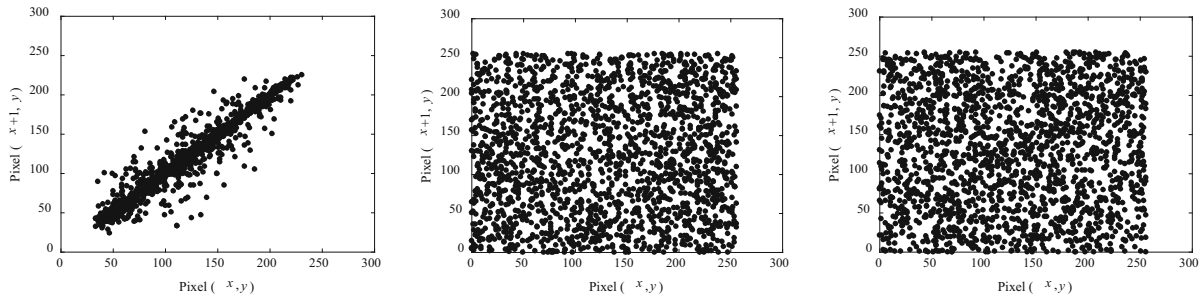


Fig. 7 Horizontal correlations of **a** plain image Lena (i.e. Fig. 3a); **b** cipher image (i.e. Fig. 4a) produced by AES-S; **c** cipher image (i.e. Fig. 5a) produced by AES-D

Table 3 Results of information entropy (unit: bit)

Histogram	Entropy
Plain images	
Fig. 6a	7.43736
Fig. 6b	7.58196
Fig. 6c	0
Fig. 6d	0
Cipher images	
Fig. 6e	7.99701
Fig. 6f	7.99706
Fig. 6g	7.99709
Fig. 6h	7.99729
Fig. 6i	7.99690
Fig. 6j	7.99679
Fig. 6k	7.99769
Fig. 6l	7.99721

images are significantly different, then the encryption system is strong sensitive to key. Similarly, for an encryption system, encrypt arbitrarily two slightly different plain images of the same size with any viable key, if the two produced cipher images are significantly different, then the encryption system is strong sensitive to plain image. In general, NPCR (number of pixels change rate) and UACI (unified average changing intensity) are used as two indicators to measure the sensitivity of image encryption system [5]. NPCR and UACI essentially reflect the difference between two images of the same size. Assuming that both $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ are size of $M \times N$, we can calculate NPCR and UACI between $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ with the following formulae.

$$\mathrm{NPCR} = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}|Sign(\boldsymbol{C}_1(i,j) - \boldsymbol{C}_2(i,j))| \times 100\% \tag{15}$$

$$\mathrm{UACI} = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\frac{|\boldsymbol{C}_1(i,j) - \boldsymbol{C}_2(i,j)|}{255-0} \times 100\% \tag{16}$$

where Sign(x) is the signal function, which returns 1 if $x > 0$, -1 if $x < 0$, or 0 if $x = 0$. For two random images of the same size, their NPCR and UACI are theoretically $255/256 \approx 99.6094\%$ and $257/768 \approx 33.4635\%$, respectively.

5.4.1 Key Sensitivity

In the key sensitivity test, take the plain images Lena, Pepper, all-black and all-white (as shown in Fig. 3a–d, respectively) as examples. In each test, randomly generate a viable key, and then slightly change the key to get a new key. Sequentially use the encryption systems (AES-S and AES-D) with these two keys to encrypt one plain image to get two cipher images, and calculate the values of NPCR and UACI between these two cipher images. Then, repeat the test for 100 times to calculate the average values of NPCR and UACI. Finally, list the results in Table 4.

As can be seen from Table 4, the calculated values of NPCR and UACI for both AES-S and AES-D systems are all fairly close to their theoretical values. So we can conclude that both AES-S and AES-D have extreme key sensitivity, and can resist the differential attacks based on the secret key.

5.4.2 Plaintext Sensitivity

In the plaintext sensitivity test, take the plain images Lena, Pepper, all-black and all-white (as shown in Fig. 3a–d, respectively) as examples. In each test, randomly generate a viable key, and select a plain image. Then slightly change the plain image to get a new plain image (e.g. change one pixel's value by 1). Sequentially use the encryption systems (AES-S and AES-D) with the key to encrypt the two plain images to get two cipher images, and calculate the values of NPCR and UACI between these two cipher images. Then, repeat the test for 100 times to calculate the average values of NPCR and UACI. Finally, list the results in Table 5.

It can be seen from Table 5, the NPCR and UACI values of AES-S are both about 50% of their theoretical values. Because the diffusion of AES-S is unidirectional from left to right, the calculated values of NPCR and UACI will close to their theoretical values if the changed pixels are in the first block of plain image, while the calculated values of NPCR and UACI will close to 0 if the changed pixels are in the last block of plain image. Generally in AES-S, the calculated values of NPCR and UACI will close to k/n of their theoretical values if the pixels in the k-th block are changed, where, n is the number of blocks, and k is the location number of the k-th block whose pixel is changed. As a result, in AES-S system, the average values of NPCR and UACI for 100 trials are about 50% of their theoretical values. This indicates that AES-S system has weak plaintext sensitivity.

As AES-D system includes a two-way diffusion, the calculated values of NPCR and UACI in AES-D system are quite close to their theoretical values, as shown in Table 5. So, AES-D system has strong plaintext sensitivity and can resist the differential attack based on chosen/known plain images.

In fact, although AES-S system has weak plaintext sensitivity, each plain block has strong plaintext sensitivity. Any pixel in each plain block changing slightly will make the produced cipher block completely different under the same secret key. This shows that AES-S system can also resist the differential attack based on chosen/known plain images.

5.5 Sensitivity Comparison Analysis

In Sect. 4, we compared the encryption/decryption speeds of both AES-S and AES-D with those of recently proposed five image cryptosystems based on chaos [16–20]. In this section, we will compare their sensitivities. It should be pointed out that we also analyzed the statistical characteristics of cipher images (such as histogram, correlation and information entropy) produced by these image cryptosystems. Since all these image cryptosystems can produce cipher images with excellent statistical characteristics, we omitted the comparative analysis of these aspects to save space.

The sensitivity indexes include NPCR and UACI. We employ the maximum relative error (MRE) to express the deviation between the index and its theoretical value. The relative error (RE) is calculated by the following

$$\mathrm{RE} = |\mathrm{INDEX}_{\mathrm{cal}} - \mathrm{INDEX}_{\mathrm{th}}|/\mathrm{INDEX}_{\mathrm{th}} \times 100\% \quad (17)$$

Table 4 Results of key sensitivity tests (unit %)

Scheme	Lena		Pepper		All-black		All-white	
	NPCR	UACI	NPCR	UACI	NPCR	UACI	NPCR	UACI
AES-S								
128-bit	99.6087	33.4561	99.6111	33.4568	99.6077	33.4673	99.6084	33.4682
192-bit	99.6092	33.4697	99.6090	33.4520	99.6068	33.4558	99.6078	33.4569
256-bit	99.6058	33.4648	99.6107	33.4663	99.6106	33.4691	99.6084	33.4603
AES-D								
128-bit	99.6070	33.4702	99.6103	33.4543	99.6135	33.4730	99.6121	33.4517
192-bit	99.6113	33.4769	99.6140	33.4713	99.6081	33.4598	99.6104	33.4647
256-bit	99.6152	33.4692	99.6112	33.4698	99.6126	33.4653	99.6063	33.4660

Table 5 Results of plaintext sensitivity tests (unit %)

Scheme	Lena		Pepper		All-black		All-white	
	NPCR	UACI	NPCR	UACI	NPCR	UACI	NPCR	UACI
AES-S								
128-bit	50.6036	17.0129	50.3975	16.9295	50.6032	16.9958	50.6051	16.9942
192-bit	51.8088	17.3980	48.4764	16.2801	51.8042	17.4051	51.8094	17.4025
256-bit	46.5362	15.6387	48.1916	16.1860	46.5369	15.6348	46.5376	15.6352
AES-D								
128-bit	99.6125	33.4743	99.6076	33.4433	99.6121	33.4708	99.6134	33.4755
192-bit	99.6079	33.4638	99.6105	33.4610	99.6123	33.4389	99.6113	33.4649
256-bit	99.6147	33.4736	99.6104	33.4676	99.6081	33.4629	99.6121	33.4584

where $INDEX_{cal}$ represents the calculated value of the index, and $INDEX_{th}$ represents the theoretical value of the index. Sequentially assign NPCR and UACI as the indexes to Eq. (17) to get the maximum relative error (MRE). The smaller the MRE is, the more close to the theoretical values the NPCR and UACI are, and the stronger the system sensitivity is.

We take the Lena (as shown in Fig. 3a) as an example to compare the key and plaintext sensitivities of AES-S, AES-D and the schemes in [16–20]. The results are listed in Table 6.

As can be seen from Table 6, AES-S has excellent key sensitivity but fair plaintext sensitivity, while AES-D has both excellent key sensitivity and excellent plaintext sensitivity (all MRE $< 0.1\%$). The scheme in [16] has average key sensitivity (MRE $> 25\%$) and good plaintext sensitivity (MRE $> 0.1\%$). The scheme in [17] has excellent key sensitivity (MRE $< 0.1\%$) but fair plaintext sensitivity (MRE $> 50\%$). The scheme in [19] has good key sensitivity (MRE $> 0.1\%$) but poor plaintext sensitivity (MRE $> 99\%$). The schemes in [18, 20] have both excellent key sensitivity and excellent plaintext sensitivity (all MRE $< 0.1\%$). So, Table 6 shows that only the schemes in [18, 20] is comparable to AES-D on sensitivities, but consider that AES-D is

Table 6 Results of sensitivity analysis (unit %)

Scheme	Key sensitivity			Plaintext sensitivity		
	NPCR	UACI	MRE	NPCR	UACI	MRE
AES-S						
128-bit	99.6087	33.4561	0.0221	50.6036	17.0129	49.1980
192-bit	99.6092	33.4697	0.0185	51.8088	17.3980	48.0090
256-bit	99.6058	33.4648	0.0039	46.5362	15.6387	53.2813
AES-D						
128-bit	99.6070	33.4702	0.0200	99.6125	33.4743	0.0323
192-bit	99.6113	33.4769	0.0400	99.6079	33.4638	0.0015
256-bit	99.6152	33.4692	0.0170	99.6147	33.4736	0.0302
Ref. [16]	77.7085	24.7620	26.0030	99.6126	33.3503	0.3383
Ref. [17]	99.6091	33.4616	0.0057	45.5548	15.3028	54.2702
Ref. [18]	99.6099	33.4624	0.0033	99.6090	33.4631	0.0012
Ref. [19]	98.0261	32.9496	1.5895	0.0015	0.0005	99.9985
Ref. [20]	99.6088	33.4623	0.0036	99.6035	33.4400	0.0702

The values of both NPCR and UACI in Table 6 are average values of 100 trials. In [17], no round operations are used. In [18], the number of rounds is 2. In [20], the number of rounds is 2 and the tiny change of secret key is 10^{-13} in key sensitivity test

much faster than the schemes in [18, 20] as shown in Table 1, AES-D is superior to all the other schemes.

6 Conclusion

In the field of information security, one of the existing viewpoints is that AES is not suitable for image encryption, and it widely accepted by most scholars. It was believed that AES is slow on image encryption when using a general-purpose computer due to image's huge volume data and high redundancy properties. However, through the researching of two image encryption schemes based on AES in CBC mode in this paper, it is confirmed that on the general-purpose computer, AES is fast on image encryption, and even faster than most of existing image encryption schemes based on chaos. In addition, this paper proposed a reasonable approach to measure the speed of image cryptosystems with the help of standard FFT algorithm. Through this work, we selected the speed of AES-D system as the lower speed limit of image cryptosystems, and the speed of AES-S system as the lower speed limit of excellent image cryptosystems. The chaotic systems based image cryptosystems which are faster than AES-D are optional solutions in practical communications.

Acknowledgements This work was fully supported by the National Science Foundation of China (Grant Nos. 61762043 and 61562035), the Natural Science Foundation of Jiangxi Province, China (Grant No. 20161BAB202058), and the Science and Technology Project of Education Department of Jiangxi Province, China (Grant No. GJJ160426).

Appendix: C Language Function for AES-Based Image Cryptosystem

The functions listed in this appendix are as shown in Table 7.

Table 7 C language functions

No.	Function name	Meanings
1	AESKey128	Generate the sub-keys (W0–W43) from the secret key of length 128 bits
2	AESKey192	Generate the sub-keys (W0–W51) from the secret key of length 192 bits
3	AESKey256	Generate the sub-keys (W0–W59) from the secret key of length 256 bits
4	AESKey	Generate the sub-keys from the secret key
5	AESRound	Fulfill one round of AES
6	AESEnc128	Complete the encryption of AES with the secret key of length 128 bits
7	AESEnc192	Complete the encryption of AES with the secret key of length 192 bits
8	AESEnc256	Complete the encryption of AES with the secret key of length 256 bits
9	AESEnc	Complete the encryption of AES
10	AESRevShiftRowAndBitRep	Inverse of both ShiftRows and SubBytes operations
11	AESRevMixColumn	Inverse of MixColumn operation
12	AESDec128	Complete the decryption of AES with the secret key of length 128 bits
13	AESDec192	Complete the decryption of AES with the secret key of length 192 bits
14	AESDec256	Complete the decryption of AES with the secret key of length 256 bits
15	AESDec	Complete the decryption of AES
16	AES_S_Enc	Fulfill the encryption of AES-S system
17	AES_S_Dec	Fulfill the decryption of AES-S system
18	AES_D_Enc	Fulfill the encryption of AES-D system
19	AES_D_Dec	Fulfill the decryption of AES-D system

```
#include "includes.h"
#define Hg 256    //The height of image
#define Wd 256    //The width of image

Int08U Sbox[256]={//S-Box, Omitted
};
Int08U RoundTblOne[256*4]={//Lookup Table for A0 in AES, Omitted
};
Int08U RoundTblTwo[256*4]={//Lookup Table for A5 in AES, Omitted
};
Int08U RoundTblThr[256*4]={//Lookup Table for A10 in AES, Omitted
};
Int08U RoundTblFou[256*4]={//Lookup Table for A15 in AES, Omitted
};
Int08U RoundTblNrOne[256]={//Lookup Table for Last round, Omitted
};
Int08U GF2p8Mul[256][256]={//Multiplication Table in GF(2^8), Omitted
};
Int08U SboxInv[256]={//Lookup Table for the Inverse of S-box, Omitted
};
/* Key of length 128-bit    */
void AESKey128(Int08U *W,Int08U *K)                    //Input: K, 16(*8);      Output: W, 44*4(*8)
{
        int i,j;
        for(i=0;i<16;i++)
        {
                W[i]=K[i];
        }
        Int08U RC[10];
        for(i=0;i<8;i++)
        {
                RC[i]=(1<<i);
        }
        RC[8]=27;RC[9]=54;
        int nr=10;
        for(i=0;i<nr;i++)
        {
                Int08U V[4];
                int idx;
                idx=(3+4*i)*4;
                V[0]=W[idx+1];V[1]=W[idx+2];V[2]=W[idx+3];V[3]=W[idx];
                W[(i+1)*16]=(Sbox[V[0]] ^ RC[i]) ^ W[i*16];
                W[(i+1)*16+1]=Sbox[V[1]] ^ W[i*16+1];
                W[(i+1)*16+2]=Sbox[V[2]] ^ W[i*16+2];
                W[(i+1)*16+3]=Sbox[V[3]] ^ W[i*16+3] ;
                for(j=0;j<12;j++)
                {
                        W[(i+1)*16+4+j]=W[i*16+4+j] ^ W[(i+1)*16+j];
                }
        }
}
void AESRound(Int08U *B,Int08U*A)
{
        Int08U Aa[16];
        Aa[0]=A[0];      Aa[1]=A[5];      Aa[2]=A[10];    Aa[3]=A[15];
                Aa[4]=A[4];      Aa[5]=A[9];      Aa[6]=A[14];    Aa[7]=A[3];
                Aa[8]=A[8];      Aa[9]=A[13];    Aa[10]=A[2];    Aa[11]=A[7];
                Aa[12]=A[12]; Aa[13]=A[1];    Aa[14]=A[6];    Aa[15]=A[11];
                for(int i=0;i<4;i++)
```

```
            {
            for(int j=0;j<4;j++)
            {
                B[i*4+j]=RoundTblOne[Aa[i*4]*4+j] ^ RoundTblTwo[Aa[i*4+1]*4+j] ^
                        RoundTblThr[Aa[i*4+2]*4+j] ^ RoundTblFou[Aa[i*4+3]*4+j];
            }
        }
    }

    void AESEnc128(Int08U *Y,Int08U *X,Int08U *W) //Input: X,W; Output: Y
    {
        int i,j;
        for(i=0;i<16;i++)
        {
            X[i]=X[i] ^ W[i];
        }
        int nr=10;
        Int08U C[16];
        for(i=1;i<nr;i++)
        {
            AESRound(&C[0],&X[0]);
            for(j=0;j<16;j++)
            {
                X[j]=C[j] ^ W[i*16+j];
            }
        }
        Y[0]=RoundTblNrOne[X[0]];Y[1]=RoundTblNrOne[X[5]];Y[2]=RoundTblNrOne[X[10]];
        Y[3]=RoundTblNrOne[X[15]];Y[4]=RoundTblNrOne[X[4]];Y[5]=RoundTblNrOne[X[9]];
        Y[6]=RoundTblNrOne[X[14]];Y[7]=RoundTblNrOne[X[3]];Y[8]=RoundTblNrOne[X[8]];
        Y[9]=RoundTblNrOne[X[13]];Y[10]=RoundTblNrOne[X[2]];Y[11]=RoundTblNrOne[X[7]];
        Y[12]=RoundTblNrOne[X[12]];Y[13]=RoundTblNrOne[X[1]];Y[14]=RoundTblNrOne[X[6]];
        Y[15]=RoundTblNrOne[X[11]];
        for(j=0;j<16;j++)
        {
            Y[j]=Y[j] ^ W[nr*16+j];
        }
    }

    void AESRevShiftRowAndBitRep(Int08U *A,Int08U *B)
    {
        Int08U Bb[16];
        Bb[0]=B[0];     Bb[1]=B[13];    Bb[2]=B[10];    Bb[3]=B[7];
        Bb[4]=B[4];     Bb[5]=B[1];     Bb[6]=B[14];    Bb[7]=B[11];
        Bb[8]=B[8];     Bb[9]=B[5];     Bb[10]=B[2];    Bb[11]=B[15];
        Bb[12]=B[12];   Bb[13]=B[9];    Bb[14]=B[6];    Bb[15]=B[3];
        for(int i=0;i<16;i++)
        {
            A[i]=SboxInv[Bb[i]];
        }
    }

    void AESRevMixColumn(Int08U *B,Int08U *C)
    {
        for(int i=0;i<4;i++)
        {
            B[i*4]   =GF2p8Mul[14][C[i*4]] ^ GF2p8Mul[11][C[i*4+1]] ^ GF2p8Mul[13][C[i*4+2]] ^
                      GF2p8Mul[9][C[i*4+3]];
            B[i*4+1] =GF2p8Mul[9][C[i*4]] ^ GF2p8Mul[14][C[i*4+1]] ^ GF2p8Mul[11][C[i*4+2]] ^
                      GF2p8Mul[13][C[i*4+3]];
            B[i*4+2] =GF2p8Mul[13][C[i*4]] ^ GF2p8Mul[9][C[i*4+1]] ^ GF2p8Mul[14][C[i*4+2]] ^
                      GF2p8Mul[11][C[i*4+3]];
            B[i*4+3] =GF2p8Mul[11][C[i*4]] ^ GF2p8Mul[13][C[i*4+1]] ^ GF2p8Mul[9][C[i*4+2]] ^
```

```
            GF2p8Mul[14][C[i*4+3]];
        }
}

void AESDec128(Int08U *Y,Int08U *X,Int08U *W) //Input: X,W; Output: Y
{
        int nr=10;
        int i,j;
        for(i=0;i<16;i++)
        {
                X[i]=X[i] ^ W[nr*16+i];
        }
        Int08U A[16];
        AESRevShiftRowAndBitRep(&A[0],&X[0]);
        for(i=nr-1;i>0;i--)
        {
                for(j=0;j<16;j++)
                {
                        A[j]=A[j] ^ W[i*16+j];
                }
                AESRevMixColumn(&X[0],&A[0]); AESRevShiftRowAndBitRep(&A[0],&X[0]);
        }
        for(i=0;i<16;i++)
        {
                Y[i]=A[i] ^ W[i];
        }
}

/* Key of length 192-bit    */
void AESKey192(Int08U *W,Int08U *K)                    //Input: K, 16(*8);      Output: W, 52*4(*8)
{
        int i,j;
        for(i=0;i<24;i++)
        {
                W[i]=K[i];
        }
        Int08U RC[8];
        for(i=0;i<8;i++)
        {
                RC[i]=(1<<i);
        }
        int nr=8;
        for(i=0;i<nr;i++)
        {
                Int08U V[4];
                int idx;
                idx=(5+6*i)*4;
                V[0]=W[idx+1];V[1]=W[idx+2];V[2]=W[idx+3];V[3]=W[idx];
                W[(i+1)*24]=(Sbox[V[0]] ^ RC[i]) ^ W[i*24];
                W[(i+1)*24+1]=Sbox[V[1]] ^ W[i*24+1];
                W[(i+1)*24+2]=Sbox[V[2]] ^ W[i*24+2];
                W[(i+1)*24+3]=Sbox[V[3]] ^ W[i*24+3];
                if(i<nr-1)
                {
                        for(j=0;j<20;j++)
                        {
                                W[(i+1)*24+4+j]=W[i*24+4+j] ^ W[(i+1)*24+j];
                        }
                }
                 else
                 {
```

```
            for(j=0;j<12;j++)
        {
            W[(i+1)*24+4+j]=W[i*24+4+j] ^ W[(i+1)*24+j];
        }
}
        }
}

void AESEnc192(Int08U *Y,Int08U *X,Int08U *W) //Input: X,W; Output: Y
{
        int i,j;
        for(i=0;i<16;i++)
        {
            X[i]=X[i] ^ W[i];
        }
        int nr=12;
        Int08U C[16];
        for(i=1;i<nr;i++)
        {
            AESRound(&C[0],&X[0]);
            for(j=0;j<16;j++)
            {
                X[j]=C[j] ^ W[i*16+j];
            }
        }
        Y[0]=RoundTblNrOne[X[0]];Y[1]=RoundTblNrOne[X[5]];Y[2]=RoundTblNrOne[X[10]];
        Y[3]=RoundTblNrOne[X[15]];Y[4]=RoundTblNrOne[X[4]];Y[5]=RoundTblNrOne[X[9]];
        Y[6]=RoundTblNrOne[X[14]];Y[7]=RoundTblNrOne[X[3]];Y[8]=RoundTblNrOne[X[8]];
        Y[9]=RoundTblNrOne[X[13]];Y[10]=RoundTblNrOne[X[2]];Y[11]=RoundTblNrOne[X[7]];
        Y[12]=RoundTblNrOne[X[12]];Y[13]=RoundTblNrOne[X[1]];Y[14]=RoundTblNrOne[X[6]];
        Y[15]=RoundTblNrOne[X[11]];
        for(j=0;j<16;j++)
        {
            Y[j]=Y[j] ^ W[nr*16+j];
        }
}

void AESDec192(Int08U *Y,Int08U *X,Int08U *W) //Input: X,W; Output: Y
{
        int nr=12;
        int i,j;
        for(i=0;i<16;i++)
        {
            X[i]=X[i] ^ W[nr*16+i];
        }
        Int08U A[16];
        AESRevShiftRowAndBitRep(&A[0],&X[0]);
        for(i=nr-1;i>0;i--)
        {
        for(j=0;j<16;j++)
        {
            A[j]=A[j] ^ W[i*16+j];
        }
        AESRevMixColumn(&X[0],&A[0]);
        AESRevShiftRowAndBitRep(&A[0],&X[0]);
}
for(i=0;i<16;i++)
{
        Y[i]=A[i] ^ W[i];
}
```

```
}

/* Key of length 256-bit    */
void AESKey256(Int08U *W,Int08U *K)                    //Input: K, 16(*8);      Output: W, 60*4(*8)
{
        int i,j;
        for(i=0;i<32;i++)
        {
                W[i]=K[i];
        }
        Int08U RC[7];
        for(i=0;i<7;i++)
        {
                RC[i]=(1<<i);
        }
        int nr=7;
        for(i=0;i<nr;i++)
        {
                Int08U V[4];
                int idx;
                idx=(7+8*i)*4;
                V[0]=W[idx+1];V[1]=W[idx+2];V[2]=W[idx+3];V[3]=W[idx];
                W[(i+1)*32]=(Sbox[V[0]] ^ RC[i]) ^ W[i*32];
                W[(i+1)*32+1]=Sbox[V[1]] ^ W[i*32+1];
                W[(i+1)*32+2]=Sbox[V[2]] ^ W[i*32+2];
                W[(i+1)*32+3]=Sbox[V[3]] ^ W[i*32+3];
                if(i<nr-1)
                {
                        for(j=0;j<28;j++)
                        {
                                if((j<12) || (j>15))
                                        W[(i+1)*32+4+j]=W[i*32+4+j] ^ W[(i+1)*32+j];
                                else
                                        W[(i+1)*32+4+j]=W[i*32+4+j] ^ Sbox[W[(i+1)*32+j]];
                        }
                }
                else
                {
                        for(j=0;j<12;j++)
                        {
                                W[(i+1)*32+4+j]=W[i*32+4+j] ^ W[(i+1)*32+j];
                        }
                }
        }
}
void AESEnc256(Int08U *Y,Int08U *X,Int08U *W) //Input: X,W; Output: Y
{
        int i,j;
        for(i=0;i<16;i++)
        {
                X[i]=X[i] ^ W[i];
        }
        int nr=14;      //The only difference compared with 192,and 128-bit
        Int08U C[16];
        for(i=1;i<nr;i++)
        {
                AESRound(&C[0],&X[0]);
                for(j=0;j<16;j++)
                {
```

```
                X[j]=C[j] ^ W[i*16+j];
            }
        }
        Y[0]=RoundTblNrOne[X[0]];Y[1]=RoundTblNrOne[X[5]];Y[2]=RoundTblNrOne[X[10]];
        Y[3]=RoundTblNrOne[X[15]];Y[4]=RoundTblNrOne[X[4]];Y[5]=RoundTblNrOne[X[9]];
        Y[6]=RoundTblNrOne[X[14]];Y[7]=RoundTblNrOne[X[3]];Y[8]=RoundTblNrOne[X[8]];
        Y[9]=RoundTblNrOne[X[13]];Y[10]=RoundTblNrOne[X[2]];Y[11]=RoundTblNrOne[X[7]];
        Y[12]=RoundTblNrOne[X[12]];Y[13]=RoundTblNrOne[X[1]];Y[14]=RoundTblNrOne[X[6]];
        Y[15]=RoundTblNrOne[X[11]];
        for(j=0;j<16;j++)
        {
            Y[j]=Y[j] ^ W[nr*16+j];
        }
}
void AESDec256(Int08U *Y,Int08U *X,Int08U *W) //Input: X,W; Output: Y
{
        int nr=14;
        int i,j;
        for(i=0;i<16;i++)
        {
            X[i]=X[i] ^ W[nr*16+i];
        }
        Int08U A[16];
        AESRevShiftRowAndBitRep(&A[0],&X[0]);
        for(i=nr-1;i>0;i--)
        {
            for(j=0;j<16;j++)
            {
                A[j]=A[j] ^ W[i*16+j];
            }
            AESRevMixColumn(&X[0],&A[0]); AESRevShiftRowAndBitRep(&A[0],&X[0]);
        }
        for(i=0;i<16;i++)
        {
            Y[i]=A[i] ^ W[i];
        }
}
void AESEnc(Int08U *Y,Int08U *X,Int08U *W,int Mode)
//Mode=128,192,256 correspond to 128,192,256-bit
{
        switch(Mode)
        {
        case 128:
            AESEnc128(&Y[0],&X[0],&W[0]); break;
        case 192:
            AESEnc192(&Y[0],&X[0],&W[0]); break;
        case 256:
            AESEnc256(&Y[0],&X[0],&W[0]); break;
        default:
            AESEnc128(&Y[0],&X[0],&W[0]); break;
        }
}
   void AESDec(Int08U *Y,Int08U *X,Int08U *W,int Mode)
   //Mode=128,192,256 correspond to 128,192,256-bit
{
        switch(Mode)
        {
        case 128:
```

```
            AESDec128(&Y[0],&X[0],&W[0]); break;
        case 192:
            AESDec192(&Y[0],&X[0],&W[0]); break;
        case 256:
            AESDec256(&Y[0],&X[0],&W[0]); break;
        default:
            AESDec128(&Y[0],&X[0],&W[0]); break;
        }
}

void AESKey(Int08U *W,Int08U *K,int Mode)          //Input: K;      Output: W
{
        switch(Mode)
        {
        case 128:
            AESKey128(&W[0],&K[0]); break;
        case 192:
            AESKey192(&W[0],&K[0]); break;
        case 256:
            AESKey256(&W[0],&K[0]); break;
        default:
            AESKey128(&W[0],&K[0]); break;
        }
}
void AES_S_Enc(Int08U *C,Int08U *P,Int08U *K,Int08U *IV,int Mode,int M,int N)
//Mode=128,192,256 correspond to 128,192,256-bit
{
        int n=M*N/16;
        Int08U W[240];
        AESKey(&W[0],&K[0],Mode);
        Int08U X[16],Y[16],XT1[16],XT2[16];
        for(int i=0;i<n;i++)
        {
            if(i==0)
            {
                for(int j=0;j<16;j++)
                {
                    X[j]=P[i*16+j] ^ IV[j]; XT2[j]=X[j];
                }
                AESEnc(&Y[0],&X[0],&W[0],Mode);
                for(int j=0;j<16;j++)
                {
                    C[i*16+j]=Y[j];
                }
                }
                else
                {
                    for(int j=0;j<16;j++)
                    {
                        X[j]=P[i*16+j] ^ C[(i-1)*16+j]; XT1[j]=X[j];
                    }
```

```
                AESEnc(&Y[0],&X[0],&W[0],Mode);
                for(int j=0;j<16;j++)
                {
                    C[i*16+j]=Y[j] ^ XT2[j];
                }
                for(int j=0;j<16;j++)
                {
                    XT2[j]=XT1[j];
                }
            }
        }
}
void AES_S_Dec(Int08U *P,Int08U *C,Int08U *K,Int08U *IV,int Mode,int M,int N)
//Mode=128,192,256 correspond to 128,192,256-bit
{
    int n=M*N/16;
    Int08U W[240];
    AESKey(&W[0],&K[0],Mode);
    Int08U X[16],Y[16],YT2[16];
    for(int i=0;i<n;i++)
    {
        if(i==0)
        {
            for(int j=0;j<16;j++)
            {
                X[j]=C[i*16+j];
            }
            AESDec(&Y[0],&X[0],&W[0],Mode);
            for(int j=0;j<16;j++)
            {
                YT2[j]=Y[j]; P[i*16+j]=Y[j] ^ IV[j];
            }
        }
        else
        {
            for(int j=0;j<16;j++)
             {
                 X[j]=C[i*16+j] ^ YT2[j];
             }
             AESDec(&Y[0],&X[0],&W[0],Mode);
             for(int j=0;j<16;j++)
             {
                 YT2[j]=Y[j]; P[i*16+j]=Y[j] ^ C[(i-1)*16+j];
             }
         }
    }
}

extern Int08U AA[Hg*Wd];
void AES_D_Enc(Int08U *C,Int08U *P,Int08U *K,Int08U *IV,int Mode,int M,int N)
//Mode=128,192,256 correspond to 128,192,256-bit
```

```
{
    int n=M*N/16;
    Int08U W[240];
    AESKey(&W[0],&K[0],Mode);
    Int08U X[16],Y[16],XT1[16],XT2[16];
    for(int i=0;i<n;i++) //Round - 1
    {
        if(i==0)
        {
            for(int j=0;j<16;j++)
            {
                X[j]=P[i*16+j] ^ IV[j]; XT2[j]=X[j];
            }
            AESEnc(&Y[0],&X[0],&W[0],Mode);
            for(int j=0;j<16;j++)
            {
                AA[i*16+j]=Y[j];
            }
        }
        else
        {
            for(int j=0;j<16;j++)
            {
                X[j]=P[i*16+j] ^ AA[(i-1)*16+j];  XT1[j]=X[j];
            }
            AESEnc(&Y[0],&X[0],&W[0],Mode);
            for(int j=0;j<16;j++)
            {
                AA[i*16+j]=Y[j] ^ XT2[j];
            }
            for(int j=0;j<16;j++)
            {
                XT2[j]=XT1[j];
            }
        }
    }
    //Round - 2
    for(int i=n-1;i>=0;i--)
    {
        if(i==n-1)
        {
            for(int j=0;j<16;j++)
            {
                X[j]=AA[i*16+j] ^ IV[j]; XT2[j]=X[j];
            }
            AESEnc(&Y[0],&X[0],&W[0],Mode);
            for(int j=0;j<16;j++)
            {
                C[i*16+j]=Y[j];
            }
        }
```

```
            else
            {
                for(int j=0;j<16;j++)
                {
                    X[j]=AA[i*16+j] ^ C[(i+1)*16+j]; XT1[j]=X[j];
                }
                AESEnc(&Y[0],&X[0],&W[0],Mode);
                for(int j=0;j<16;j++)
                {
                    C[i*16+j]=Y[j] ^ XT2[j];
                }
                for(int j=0;j<16;j++)
                {
                    XT2[j]=XT1[j];
                }
            }
        }
}

extern Int08U BB[Hg*Wd];
void AES_D_Dec(Int08U *P,Int08U *C,Int08U *K,Int08U *IV,int Mode,int M,int N)
//Mode=128,192,256 correspond to 128,192,256-bit
{
    int   n=M*N/16;
    Int08U   W[240];
    AESKey(&W[0],&K[0],Mode);
    Int08U X[16],Y[16],YT2[16];
    for(int i=n-1;i>=0;i--) //Round - 1
    {
        if(i==n-1)
        {
            for(int j=0;j<16;j++)
            {
                X[j]=C[i*16+j];
            }
            AESDec(&Y[0],&X[0],&W[0],Mode);
            for(int j=0;j<16;j++)
            {
                YT2[j]=Y[j];BB[i*16+j]=Y[j] ^ IV[j];
            }
        }
        else
        {
            for(int j=0;j<16;j++)
            {
                X[j]=C[i*16+j] ^ YT2[j];
            }
            AESDec(&Y[0],&X[0],&W[0],Mode);
            for(int j=0;j<16;j++)
            {
                YT2[j]=Y[j];BB[i*16+j]=Y[j] ^ C[(i+1)*16+j];
            }
        }
    }
    //Round - 2
    for(int i=0;i<n;i++)
```

```
        {
            if(i==0)
            {
                for(int j=0;j<16;j++)
                {
                    X[j]=BB[i*16+j];
                }
                AESDec(&Y[0],&X[0],&W[0],Mode);
                for(int j=0;j<16;j++)
                {

                YT2[j]=Y[j];P[i*16+j]=Y[j] ^ IV[j];
            }
        }
        else
        {
            for(int j=0;j<16;j++)
            {
                X[j]=BB[i*16+j] ^ YT2[j];
            }
            AESDec(&Y[0],&X[0],&W[0],Mode);
            for(int j=0;j<16;j++)
            {
                YT2[j]=Y[j]; P[i*16+j]=Y[j] ^ BB[(i-1)*16+j];
            }
        }
    }
}
```

References

1. Westlund, H. B. (2002). NIST reports measurable success of advanced encryption standard. *Journal of Research of the National Institute of Standards and Technology, 107*(3), 307.
2. Daemen, J., & Rijmen, V. (2001). The design of Rijndael. *Information Security & Cryptography, 26*(3), 137–139.
3. Daemen, J., & Rijmen, V. (2002). *The design of Rijndael AES-the advanced encryption standard*. Berlin: Springer-Verlag.
4. Zhang, Y., Li, X., & Hou, W. (2017). A fast image encryption scheme based on AES. *Proceedings of ICIVC, 2,* 624–2628.
5. Chen, G., Mao, Y., & Chui, C. K. (2004). A symmetric image encryption scheme based on 3D chaotic cat maps. *Chaos, Solitons & Fractals, 21*(3), 749–761.
6. Pareek, N. K., Patidar, V., & Sud, K. K. (2006). Image encryption using chaotic logistic map. *Image and Vision Computing, 24*(9), 926–934.
7. Gao, T., & Chen, Z. (2008). A new image encryption algorithm based on hyper-chaos. *Physics Letters A, 372*(4), 394–400.
8. Song, C. Y., Qiao, Y. L., & Zhang, X. Z. (2013). An image encryption scheme based on new spatiotemporal chaos. *Optik, 124*(18), 3329–3334.
9. Wang, X., Teng, L., & Qin, X. (2012). A novel colour image encryption algorithm based on chaos. *Signal Processing, 92*(4), 1101–1108.
10. Liu, Z., Zhang, Y., Li, S., Liu, W., Liu, W., Wang, Y., et al. (2013). Double image encryption scheme by using random phase encoding and pixel exchanging in the gyrator transform domains. *Optics & Laser Technology, 47*(1), 152–158.
11. Liu, Z., Li, S., Liu, W., Liu, W., & Liu, S. (2013). Image hiding scheme by use of rotating squared sub-image in the gyrator transform domains. *Optics & Laser Technology, 45*(1), 198–203.
12. Li, C., Liu, Y., Zhang, L., & Wong, K. W. (2014). Cryptanalyzing a class of image encryption schemes based on Chinese remainder theorem. *Signal Processing: Image Communication, 29*(8), 914–920.
13. Liu, Y., Zhang, L., Zhang, Y., & Wong, K. W. (2015). Chosen-plaintext attack of an image encryption scheme based on modified permutation–diffusion structure. *Nonlinear Dynamics, 84*(4), 2241–2250.
14. Zeng, L., Liu, R., Zhang, L., & Wong, K. W. (2016). Cryptanalyzing an image encryption algorithm based on scrambling and Veginère cipher. *Multimedia Tools & Applications, 75*(10), 5439–5453.
15. Zhang, L., Liu, Y., Wong, K. W., Pareschi, F., Zhang, Y., & Setti, G. (2017). On the security of a class of diffusion mechanisms for image encryption. *IEEE Transactions on Cybernetics, 99,* 1–13.
16. Hua, Z., & Zhou, Y. (2016). Image encryption using 2D Logistic-adjusted-Sine map. *Information Sciences, 339,* 237–253.
17. Eslami, Z., & Bakhshandeh, A. (2013). An improvement over an image encryption method based on total shuffling. *Optics Communications, 286*(1), 51–55.
18. Cheng, P., Yang, H., Wei, P., & Zhang, W. (2015). A fast image encryption algorithm based on chaotic and lookup table. *Nonlinear Dynamics, 79*(3), 2121–2131.

19. Ünal Çavuşoğlu, S., Kaçar, S., Pehlivan, I., & Zengin, A. (2017). Secure image encryption algorithm design using a novel chaos based S-Box. *Chaos, Solitons & Fractals, 95,* 92–101.
20. Ye, G., Zhao, H., & Chai, H. (2016). Chaotic image encryption algorithm using wave-line permutation and block diffusion. *Nonlinear Dynamics, 83*(4), 2067–2077.
21. Zhu, H., Zhang, X., Yu, H., Zhao, C., & Zhu, Z. (2016). A novel image encryption scheme using the composite discrete chaotic system. *Entropy, 18*(8), 276.
22. Chen, J., Zhu, Z., Fu, C., Yu, H., & Zhang, L. (2015). A fast chaos-based image encryption scheme with a dynamic state variables selection mechanism. *Communications in Nonlinear Science and Numerical Simulation, 20*(3), 846–860.
23. Wang, X., Liu, L., & Zhang, Y. (2015). A novel chaotic block image encryption algorithm based on dynamic random growth technique. *Optics and Lasers in Engineering, 66*(66), 10–18.
24. Assad, S. E., & Farajallah, M. (2016). A new chaos-based image encryption system. *Signal Processing Image Communication, 41,* 144–157.
25. Zhang, X., Fan, X., Wang, J., & Zhao, Z. (2016). A chaos-based image encryption scheme using 2D rectangular transform and dependent substitution. *Multimedia Tools & Applications, 75*(4), 1745–1763.
26. Luo, Y., Cao, L., Qiu, S., Lin, H., Harkin, J., & Liu, J. (2016). A chaotic map-control-based and the plain image-related cryptosystem. *Nonlinear Dynamics, 83*(4), 2293–2310.
27. Farajallah, M., & Assad, S. E. (2016). Fast and secure chaos-based cryptosystem for images. *International Journal of Bifurcation & Chaos, 26*(2), 1650021.
28. Tong, X., Zhang, M., Wang, Z., & Ma, J. (2016). A joint color image encryption and compression scheme based on hyper-chaotic system. *Nonlinear Dynamics, 84*(4), 2333–2356.
29. Xu, H., Tong, X., & Meng, X. (2016). An efficient chaos pseudo-random number generator applied to video encryption. *Optik, 127*(20), 9305–9319.
30. Fawaz, Z., Noura, H., & Mostefaoui, A. (2016). An efficient and secure cipher scheme for images confidentiality preservation. *Signal Processing: Image Communication, 42,* 90–108.
31. Chai, X., Chen, Y., & Broyde, L. (2017). A novel chaos-based image encryption algorithm using DNA sequence operations. *Optics and Lasers in Engineering, 88,* 197–213.
32. Chai, X., Gan, Z., Yuan, K., Lu, Y., & Chen, Y. (2017). An image encryption scheme based on three-dimensional Brownian motion and chaotic system. *Chinese Physics B, 26*(2), 020504.
33. Shannon, C. E. (1998). Communication theory of secrecy systems, M.d.computing Computers in Medical. *Practice, 15*(1), 57–116.
34. Li, S., Li, C., Chen, G., Bourbakis, N. G., & Lo, K.-T. (2008). A general quantitative cryptanalysis of permutation-only multimedia ciphers against plaintext attacks. *Signal Processing: Image Communication, 23*(3), 212–223.
35. Jolfaei, A., Wu, X., & Muthukkumarasamy, V. (2016). On the security of permutation-only image encryption schemes. *IEEE Transactions on Information Forensics and Security, 11*(2), 235–246.
36. Frigo, M., & Johnson, S. G. (2005). The design and implementation of FFTW3. *Proceedings of the IEEE, 93*(2), 216–231.