Applied Soft Computing 11 (2011) 514–522

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A new chaos-based fast image encryption algorithm

Yong Wang[a,b,*], Kwok-Wo Wong[b], Xiaofeng Liao[c], Guanrong Chen[b]

[a] Key Laboratory of Electronic Commerce and Logistics of Chongqing, Chongqing University of Posts and Telecommunications, Chongqing 400065, China
[b] Department of Electronic Engineering, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon Tong, Hong Kong, China
[c] Department of Computer Science and Engineering, Chongqing University, Chongqing 400044, China

ARTICLE INFO

Article history:
Received 18 October 2008
Received in revised form
19 November 2009
Accepted 6 December 2009
Available online 16 December 2009

Keywords:
Spatiotemporal chaos
Image encryption
Cryptography
Information security

ABSTRACT

In recent years, various image encryption algorithms based on the permutation–diffusion architecture have been proposed where, however, permutation and diffusion are considered as two separate stages, both requiring image-scanning to obtain pixel values. If these two stages are combined, the duplicated scanning effort can be reduced and the encryption can be accelerated. In this paper, a fast image encryption algorithm with combined permutation and diffusion is proposed. First, the image is partitioned into blocks of pixels. Then, spatiotemporal chaos is employed to shuffle the blocks and, at the same time, to change the pixel values. Meanwhile, an efficient method for generating pseudorandom numbers from spatiotemporal chaos is suggested, which further increases the encryption speed. Theoretical analyses and computer simulations both confirm that the new algorithm has high security and is very fast for practical image encryption.

1. Introduction

With the rapid growth of image transmission through computer networks especially the Internet, the security of digital images has become a major concern. Image encryption, in particular, is urgently needed but it is a challenging task—it is quite different from text encryption due to some intrinsic properties of images such as bulky data capacity and high redundancy, which are generally difficult to handle by using traditional techniques. Nevertheless, many new image encryption schemes have been suggested in recent years, among which the chaos-based approach appears to be a promising direction [1–10].

A general permutation–diffusion architecture for chaos-based image encryption was employed in Refs. [1,11–15] as illustrated in Fig. 1. In the permutation stage, the image pixels are relocated but their values remain unchanged. In the diffusion stage, the pixel values are modified so that a tiny change in one-pixel spreads out to as many pixels as possible. Permutation and diffusion are two separate and iterative stages, and they both require scanning the image in order to obtain the pixel values. Thus, in the encryption process, each round of the permutation–diffusion operation requires at least twice scanning the same image. This effort is actually duplicated but may be avoided if the permutation and diffusion operations can be combined, *i.e.*, via changing the values of the pixels while relocating them, as illustrated in Fig. 2. As a result, the image only needs to be scanned once so that the encryption speed and efficiency is significantly improved.

On the other hand, spatiotemporal chaos has attracted more and more interests among researchers in the fields of mathematics, physics and engineering. Compared with simple chaotic maps, this kind of spatiotemporal chaos possesses two additional merits for cryptographic purposes. First, observe that due to the finite computing precision, orbits of temporal discrete chaotic systems will eventually become periodic. However, the period of spatiotemporal chaos is found much longer than that of temporal chaotic maps [16] so that the periodicity problem is practically avoided [17]. Second, a spatiotemporal chaotic system is high-dimensional, having a number of positive Lyapunov exponents that guarantee the complex dynamical behavior or high randomness. It is therefore more difficult to predict the time series generated by this kind of chaotic systems.

In this paper, an image encryption algorithm with the architecture of combining permutation and diffusion is proposed. The plain-image is first partitioned into blocks of 8×8 pixels. A spatiotemporal chaotic system is then employed to generate the pseudorandom sequence used for diffusing and shuffling the blocks. The objectives of this new design include: (i) to efficiently extract good pseudorandom sequences from a spatiotemporal chaotic system and (ii) to simultaneously perform permutation and diffusion operations for fast encryption.

The rest of this paper is organized as follows: Section 2 focuses on the efficient generation of pseudorandom sequences from spa-

* Corresponding author at: Key Laboratory of Electronic Commerce and Logistics of Chongqing, Chongqing University of Posts and Telecommunications, Chongqing 400065, China. Fax: +86 23 62461172.
E-mail address: wangyong_cqupt@163.com (Y. Wang).

1568-4946/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2009.12.011

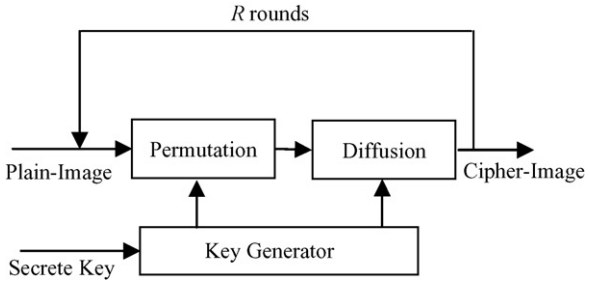

Fig. 1. Image cryptosystem based on the permutation and diffusion operations.

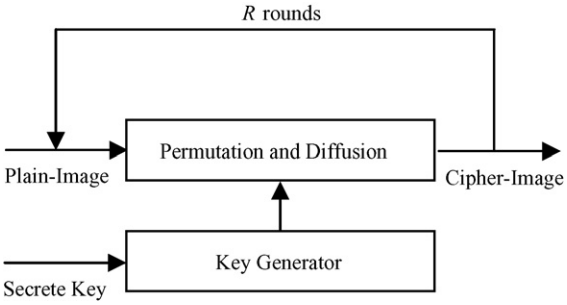

Fig. 2. Image cryptosystem combining the permutation–diffusion architecture.

tiotemporal chaos. In Section 3, the proposed algorithm is described in detail. Section 4 presents simulation results and performance analyses. In Section 5, conclusions are drawn.

2. Pseudorandom sequences generated from spatiotemporal chaos

2.1. Approach to obtaining pseudorandom numbers

A general nearest-neighboring spatiotemporal chaos system, also called nearest-neighboring coupled-map lattices (NCML) [18], can be described by

$$x_{n+1}(i) = (1-\varepsilon)f(x_n(i)) + \varepsilon f(x_n(i+1)) \quad (1)$$

where $n = 1, 2, \ldots$, is the time index; $i = 1, 2, \ldots, N$, is the lattice state index; f is a chaotic map, and $\varepsilon \in (0, 1)$ is a coupling constant. The periodic boundary condition $x_n(N+i) = x_n(i)$ is imposed into this system. Moreover, the tent map is chosen as the local chaotic map, given by

$$x_{i+1} = \begin{cases} x_i/b, & x_i \le b \\ (1-x_i)/(1-b), & x_i > b \end{cases} \quad (2)$$

where $b \in (0, 1)$ is a constant. Here, N is chosen as 8, while the parameters are selected as $\varepsilon = 0.05$ and $b = 0.4999$ in order to have good chaotic properties [19,20].

The traditional approach to extracting pseudorandom numbers from the output of a chaotic system involves iterating the chaotic map and then extracting a value from its current state variable. These two operations are performed repeatedly until sufficient pseudorandom numbers are obtained. For the NCML, the local chaotic map of each lattice is first iterated. Then, the new state values are calculated according to the coupling relationship between the lattices. Obviously, iterating the NCML requires much more computational effort than iterating a simple chaotic map. The efficiency will be very low if only one value is extracted from the NCML at each time. Therefore, to improve the efficiency, more numbers should be extracted in each iteration.

Table 1
Time required for 10,000,000 runs of various basic operations performed on *PC*1 and *PC*2.

Operators	Time required (ms)	
	*PC*1	*PC*2
And ($\wedge$)	15	12
Complement ($\neg$)	15	12
Exclusive OR ($\oplus$)	16	12
Inclusive OR ($\vee$)	16	12
Modulus (mod)	16	12
Addition (+)	18	16
Multiplication ($\times$)	63	47
Converting floating-point to integer	210	160

The state value of a chaotic map is a floating-point number, but a pseudorandom number in the form of an integer is usually required. This means that the conversion from floating-points to integers cannot be avoided in practical applications. However, it is found from computer simulations that such a conversion is time-consuming. To justify this, various basic operations are repeated for 10,000,000 times on two personal computers with different configurations, where one is named as *PC*1 with a 1.3 GHz Pentium processor and 256 M RAM while the other is called *PC*2 with a 2.67 GHz Pentium D processor and 1G RAM. The timing data listed in Table 1 indicate that multiplication and conversion from floating-points to integers should be avoided in order to have high efficiency of generating pseudorandom numbers.

Based on the above analysis, the following steps for generating 64 pseudorandom numbers from the lattice values are suggested.

Step 1. Iterate the NCML once and extract 16 bits (9th to 24th bits after the decimal point) from each lattice value. Thus, a total of 128 bits are obtained. Divide these bits into 16 bytes and denote them as $A_0, A_1, \ldots, A_{15}$.

Step 2. Generate 16 numbers, $S_0, S_1, \ldots, S_{15}$ from $A_0, A_1, \ldots, A_{15}$ according to the following formula:

$$\begin{array}{llll} S_0 = F(A_0, A_1, A_2, A_3); & S_1 = G(A_1, A_2, A_3, A_0); & S_2 = H(A_2, A_3, A_0, A_1); & S_3 = I(A_3, A_0, A_1, A_2) \\ \vdots & \vdots & \vdots & \vdots \\ S_{12} = F(A_{12}, A_{13}, A_{14}, A_{15}); & S_{13} = G(A_{13}, A_{14}, A_{15}, A_{12}); & S_{14} = H(A_{14}, A_{15}, A_{12}, A_{13}); & S_{15} = I(A_{15}, A_{12}, A_{13}, A_{14}) \end{array} \quad (3)$$

where functions F, G, H, I are defined as

$$F(a, b, c, d) = \{[(a \wedge b) \vee ((\neg a) \wedge c)] + d\} \bmod 256 \quad (4)$$

$$G(a, b, c, d) = \{[(a \wedge c) \vee (b \wedge (\neg c))] + d\} \bmod 256 \quad (5)$$

$$H(a, b, c, d) = [(a \oplus b \oplus c) + d] \bmod 256 \quad (6)$$

$$I(a, b, c, d) = \{[b \oplus (a \vee (\neg c))] + d\} \bmod 256 \quad (7)$$

Step 3. Change the values of $A_0, A_1, \ldots, A_{15}$ as

$$A_0 \to A_1,\ A_1 \to A_2,\ \ldots,\ A_{14} \to A_{15},\ A_{15} \to A_0 \quad (8)$$

If 64 pseudorandom numbers have already been generated, go to *Step* 4; otherwise, go to *Step* 2 to generate the next 16 pseudorandom numbers.

Step 4. Substitute the 64 numbers according to the S-box of AES [21], which are expressed in hexadecimal form in Fig. 3.

Step 5. Generate the 64 output numbers according to

$$R_i = \{[S_b(i) \oplus S_b((i+1) \bmod 64)] + [S_b((i+2) \bmod 64) \oplus S_b((i+3) \bmod 64)]\} \bmod 256 \quad (9)$$