Available online at www.sciencedirect.com
ScienceDirect
Information Sciences 178 (2008) 1391–1406
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

One-way hash function construction based on 2D coupled map lattices

Yong Wang [a,b,*], Xiaofeng Liao [a], Di Xiao [a], Kwok-Wo Wong [c]

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

Received 15 December 2006; received in revised form 7 October 2007; accepted 11 October 2007

Abstract

An algorithm for constructing one-way hash function based on spatiotemporal chaos is proposed. A two-dimensional coupled map lattices (2D CML) with parameters leading to the largest Lyapunov exponent is employed. The state of the 2D CML is dynamically determined by its previous state and the message bit at the corresponding positions. The hash value is obtained by a linear transform on the final state of the 2D CML. Theoretical analysis and computer simulation indicate that our algorithm has good statistical properties, strong collision resistance and high flexibility. It is practical and reliable, with high potential to be adopted as a strong hash function for providing data integrity.

Keywords: Hash function; Two-dimensional coupled map lattices; Spatiotemporal chaos

1. Introduction

A hash function is a fundamental building block of information security and plays an important role in modern cryptography. It takes a message as input and produces an output referred to as a hash value, or simply *hash*. More precisely, a hash function h maps bit strings of arbitrary finite length to strings of fixed length. Generally, hash functions can be classified into two categories [10,13]: unkeyed hash functions with a single input parameter (a message) for data integrity, and keyed hash functions, usually known as message authentication code (MAC), with two distinct inputs. Conventional hash functions, such as MD5 and SHA, involve logical operations or multi-round iterations of some available ciphers. Although each step of the former is simple, the number of processing rounds could be huge even if the message is very short. The security and efficiency of the latter totally rely on the intrinsic cipher, which needs complicated computations. Moreover, recent investigations on the collision frequencies reveal many undiscovered flaws in the well-known methods,

* Corresponding author. Address: Department of Computer Science and Engineering, Chongqing University, Chongqing 400044, PR China. Tel.: +86 23 62784654.
E-mail address: wangyong_cqupt@163.com (Y. Wang).

0020-0255/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2007.10.008

such as MD5, SHA1, and RIPEMD [20–22]. As a result, the research on the design of secure and efficient keyed/unkeyed hash functions attracts more and more attentions.

As a ubiquitous phenomenon in nature, chaos is a kind of deterministic random-like process found in nonlinear dynamical systems. It is employed for data protection due to its attractive features such as sensitive to initial value, random-like and ergodic [6]. For example, discretized chaotic maps are used in encryption algorithms [5,12,25,31] and security protocols [2,26] while coupled map lattices are adopted to construct a stream cipher [7]. Being considered as a novel direction in the construction of hash functions, chaos-based approaches have been attracting more and more research interests [9,24,27–29]. Like chaotic cryptosystems, chaos-based hash functions utilize some interesting characteristics of chaos, such as sensitivity to initial conditions, ergodicity and mixing property. Based on Baptista's encryption method, Wong developed a scheme combining encryption and hashing [24]. Although it is able to encrypt messages and generate the corresponding hash value simultaneously, the efficiency and security of this scheme need further improvements [1]. Based on the piecewise linear chaotic map (PWLCM) or tent map, a number of hash algorithms are proposed [9,27,28]. Their efficiency is high, especially the one suggested in [27]. However, Li et al. pointed out in [8] that the uniform distribution property of PWLCM collapses under finite computing precision despite it is valid in continuous phase space. Moreover, methods for predicting chaotic time series have been suggested [11,17,18]. To prevent attackers from breaking the hash function by predicting the chaotic series, complex chaotic map should be employed. A method for constructing hash functions based on spatiotemporal chaos was proposed [29]. Although the chaotic series is difficult to predict, the efficiency of this method is not high enough, especially when the message is long. Recently, a keyed hash function is proposed [30]. Owing to making full use of the nth-order chaotic system, which is a complex system with good cryptographic properties, this algorithm satisfies the performance requirement of keyed hash functions and is easy to be implemented by hardware for its filter-based structure. All in all, further studies on complex chaotic systems and chaos-based hash functions are needed.

Recently, spatiotemporal chaos has been attracting more and more interests among researchers in the fields of mathematics, physics, and engineering. Compared with simple chaotic maps, spatiotemporal chaos possesses two additional merits for cryptographic purpose. Due to the finite computing precision, chaotic orbits will eventually become periodic. The period of spatiotemporal chaos is longer than that of simple chaotic maps [19]. In particular, the period of chaotic orbits generated by a system with a large number of chaotic coupled oscillators is too long to be reached in practical communications. Therefore, periodicity can be practically avoided in spatiotemporal chaotic systems [7]. Moreover, spatiotemporal chaotic system is a high-dimensional chaotic system, which has a number of positive Lyapunov exponents that guarantee complex dynamical behavior. It is more difficult or even impossible to predict the time series generated by spatiotemporal chaos.

In this paper, an unkeyed algorithm for one-way hash function construction based on a spatiotemporal chaotic system is proposed. A 2D CML formed by logistic maps is used as the spatiotemporal chaotic system. Each 8-bit message is mapped to a number in the interval $[0, 1]$ by a linear transform. The transformation output is then fed into the 2D CML for iteration. The cipher block chaining mode (CBC) is adopted to ensure that the state of the 2D CML is jointly determined by the previous iteration output and the 8-bit message. The final hash value is obtained by means of a linear transformation on the output of the 2D CML. Theoretical analysis and computer simulation indicate that the proposed hash function possesses the advantages of irreversibility, weak collision and flexibility. Moreover, it is secure and easy to be implemented, with high potential to be adopted for providing data integrity.

The remaining part of the paper is organized as follows. The feasibility of hash function construction based on spatiotemporal chaos is analyzed in Section 2. Section 3 is devoted to the design of hash function based on 2D CML. Performance of the proposed hash function is analyzed in Section 4. Finally, conclusions are drawn in Section 5.

2. Feasibility of hash function construction based on 2D CML

A one-way function h is a function that for each x in the domain of h, it is easy to compute $h(x)$; but for essentially all y in the range of h, it is computationally infeasible to find any x such that $y = h(x)$. Hash function is a special kind of one-way function that possesses the following properties [13]:

(1) *Compression*: h maps an input x of arbitrary finite length to an output $h(x)$ of fixed length n.
(2) *Irreversibility*: Given h and an input x, it is easy to compute $h(x)$. However, it is computationally infeasible to find any input which hashes to a specific output, i.e., to find any pre-image x such that $h(x) = y$ when given any y for which a corresponding input is not known.
(3) *Second pre-image resistance*: It is computationally infeasible to find any second input which has the same output as any specific input, i.e., given x, to find a second pre-image $x' \neq x$ such that $h(x) = h(x')$.
(4) *Sensitivity to input bits*: Each output bit is related to input bits. An avalanche property similar to that of good block ciphers is desirable whereby every input bit affects every output bit.

It is well known that chaos has the following properties: sensitivity to tiny changes in initial conditions and parameters, mixing, and ergodicity, etc. Furthermore, the iteration process is one-way and iterations result in a totally different orbit. The 2D CML is a high-dimensional chaotic system, which not only possesses the properties of chaos, but also leads to a more complicated chaotic time series and less sensitive to finite computing precision. The inherent merits of 2D CML constitute the solid theoretical foundation for the construction of excellent hash functions.

3. A one-way hash function based on 2D CML

3.1. The 2D CML model

A general nearest-neighbor 2D CML can be described as

$$x_{n+1}^{i,j} = (1-\varepsilon) f(x_n^{i,j}) + \frac{\varepsilon}{4}[f(x_n^{i+1,j}) + f(x_n^{i-1,j}) + f(x_n^{i,j+1}) + f(x_n^{i,j-1})], \tag{1}$$

where $n = 1, 2, \ldots$ is the time index or state index; $i = 1, 2, \ldots, M$ is the lattice row index; $j = 1, 2, \ldots, N$ is the lattice column index; f is a local chaotic map and $\varepsilon \in (0, 1)$ is a coupling constant. To reduce the computational complexity, we use the following model to construct hash function:

$$x_{n+1}^{i,j} = (1-\varepsilon) f(x_n^{i,j}) + \frac{\varepsilon}{2}[f(x_n^{i+1,j}) + f(x_n^{i,j+1})]. \tag{2}$$

The periodic boundary conditions, $x_n^{M+i,j} = x_n^{i,j}$ and $x_n^{i,N+j} = x_n^{i,j}$, are used in the 2D CML. Here the logistic map is taken as the local map, given as

$$f(x) = \mu x(1-x), \tag{3}$$

where $\mu \in (3.57, 4)$ is a constant.

It is well known that the system is chaotic if the largest Lyapunov exponent (LLE) is greater than 0. The larger the LLE is, the more chaotic the system is. Since μ, ε, M, N may affect the LLE, only one parameter is varied each time with the remaining three parameters fixed, so as to observe the influence of the varying parameter on the LLE [23]. The LLEs of 2D CML with various μ, ε, M or N are shown in Figs. 1–4, respectively.

According to the result of numerical analysis, some conclusions are drawn below:

(i) The LLE increases as the parameter μ increases.
(ii) The parameter ε only has a little influence on LLE. When $\mu = 4$, the LLE firstly decreases with ε, then remains nearly constant and finally increases slightly.
(iii) When $M \geqslant 4$ and $N \geqslant 8$, there is almost no effect on the LLE as the size of 2D CML increases.

3.2. Algorithm for generating the hash

Based on the conclusion drawn above, we take $\mu = 4$, $\varepsilon = 0.05$, $M = 4$, $N = 8$ in Eq. (2), that is

$$x_{n+1}^{i,j} = 3.8 x_n^{i,j}(1 - x_n^{i,j}) + 0.1[x_n^{i+1,j}(1 - x_n^{i+1,j}) + x_n^{i,j+1}(1 - x_n^{i,j+1})], \tag{4}$$

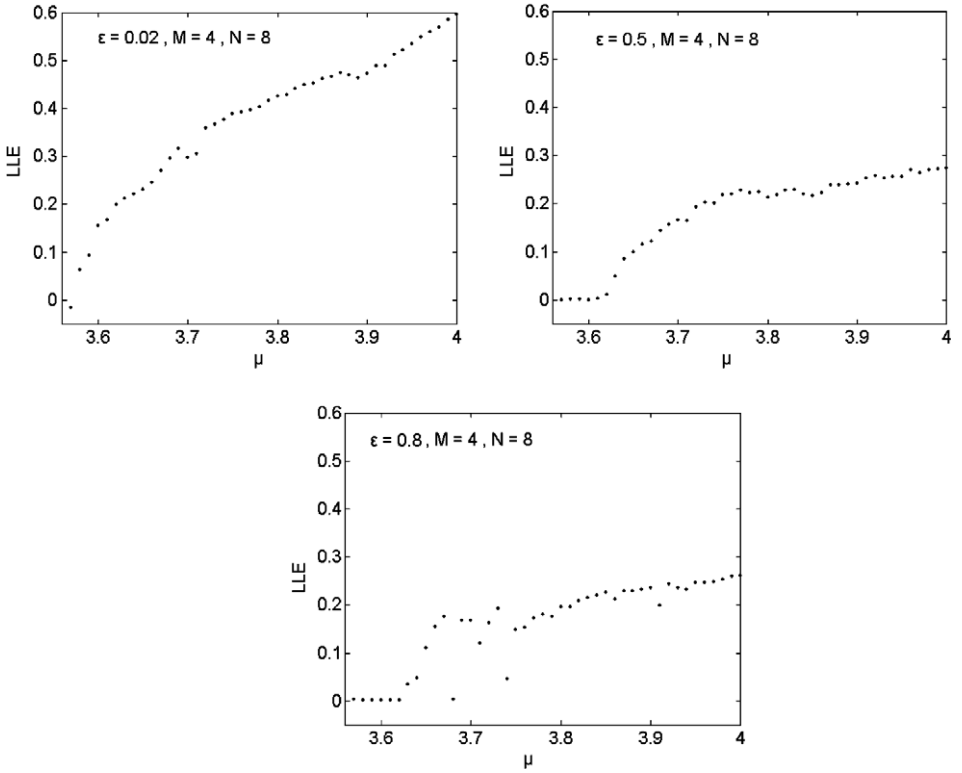

Fig. 1. LLEs with various μ.

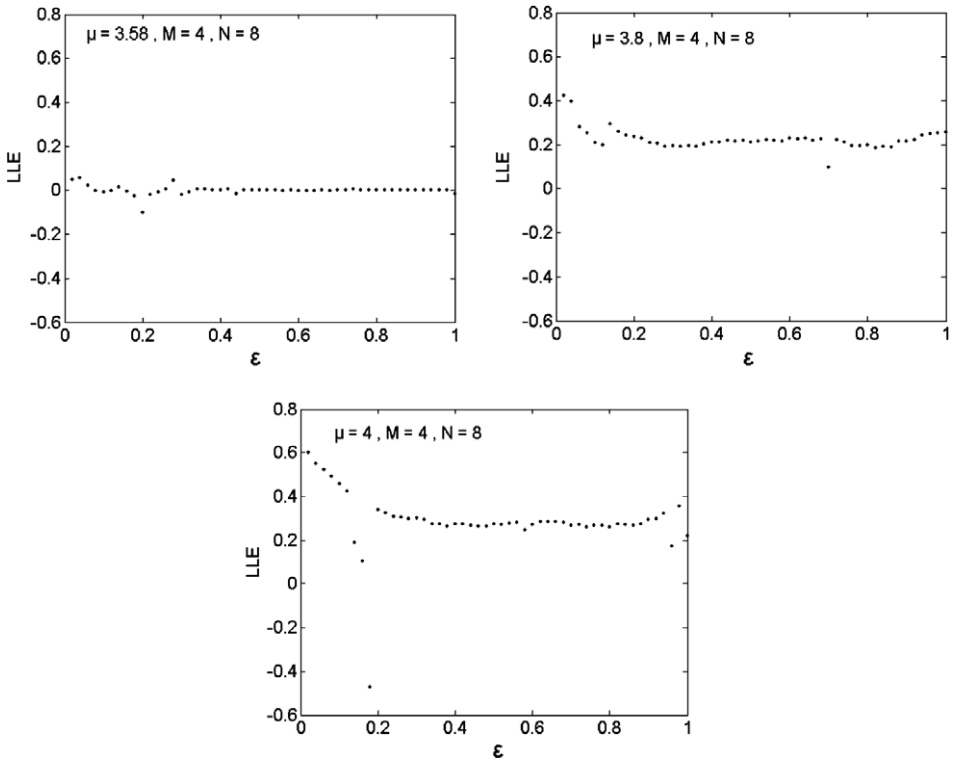

Fig. 2. LLEs with various ε.

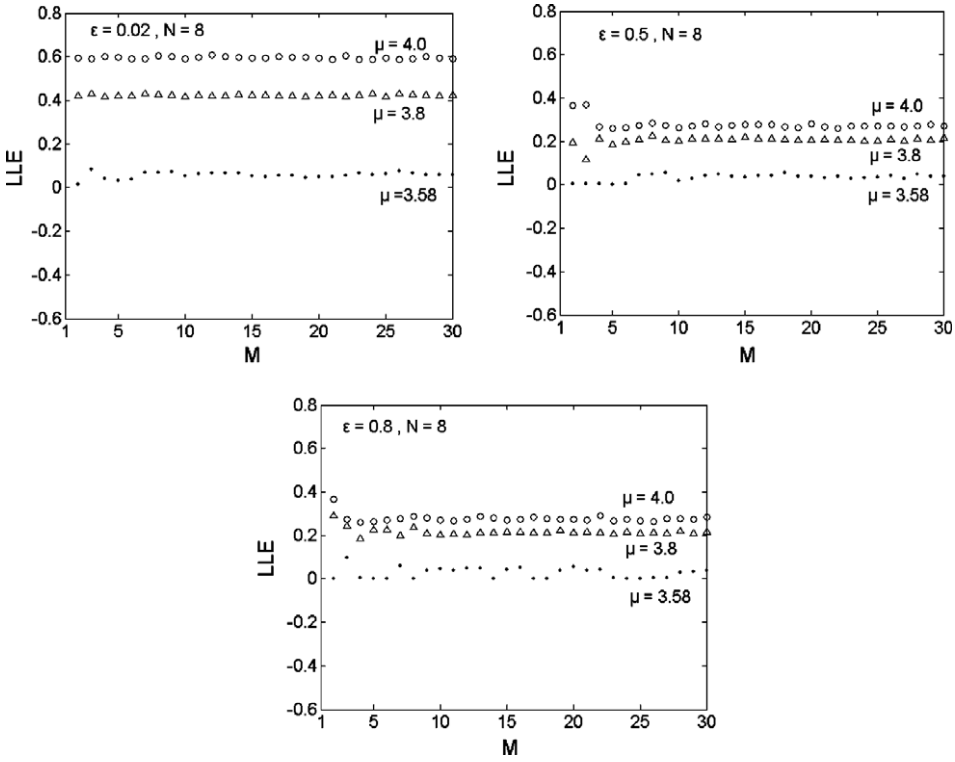

Fig. 3. LLEs with various M.

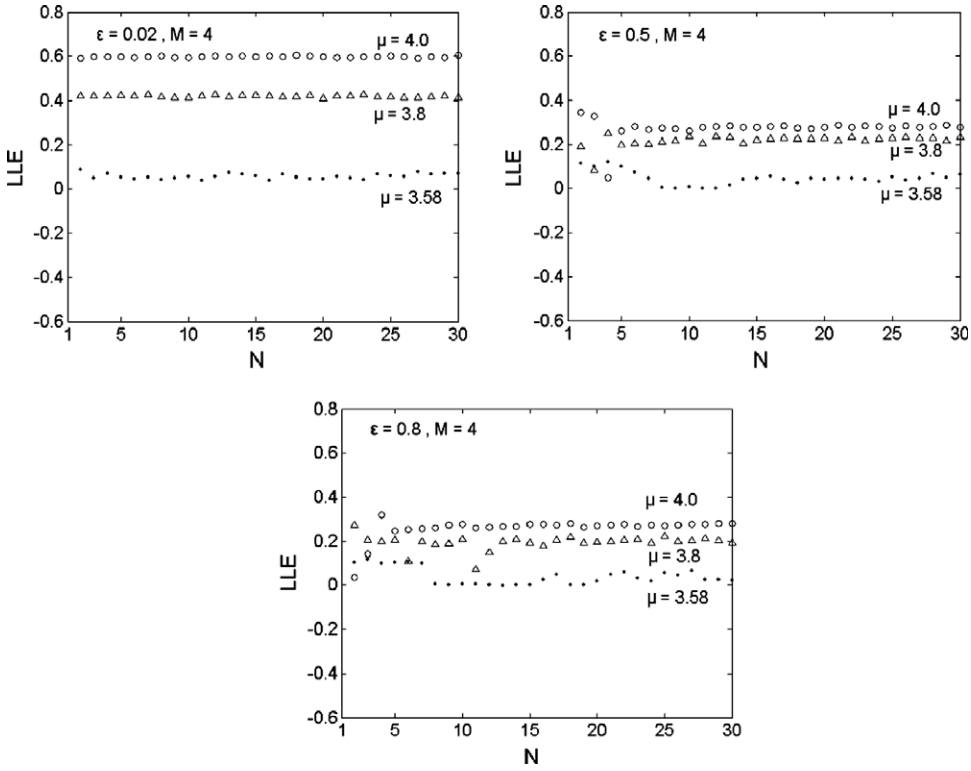

Fig. 4. LLEs with various N.

where $i = 1, 2, \ldots, 4$; $j = 1, 2, \ldots, 8$. The periodic boundary conditions are $x_n^{4+i,j} = x_n^{i,j}$ and $x_n^{i,8+j} = x_n^{i,j}$. The detailed algorithm for generating the hash is described as follows:

INPUT: bit string y of arbitrary length
OUTPUT: 128-bit hash value

(1) *Definition of constants.* Define sixteen 8-bit initial values (IVs) which are expressed in hexadecimal format, that is: IV[1...16] = [01, 23, 45, 67, 89, AB, CD, EF, FE, DC, BA, 98, 76, 54, 32, 10]. Then set

$$x_1^{1,j} = (IV[j] + 0.8)/256, \quad x_1^{3,j} = (IV[8 + j] + 0.8)/256, \tag{5}$$

where $j = 1, 2, \ldots, 8$.

(2) *Preprocessing*. Partition bit string y into a number of 128-bit blocks. Append some zero bits to the last block if necessary. Then translate the partitioned y to the corresponding ASCII number and map them into array C whose elements are numbers in the interval $(0, 1)$. This mapping is achieved by a linear transform

$$C[i] = (A[i] + 0.8)/256, \tag{6}$$

where $A[i]$ is the ith ASCII number of y and $C[i]$ is the ith element of array C. Divide the array C into R groups, each group consists of l elements (here $l = 16$):

$$C = \underbrace{C[1]C[2], \ldots, C[l]}_{G_1}, \underbrace{C[l+1], \ldots, C[2l]}_{G_2}, \ldots, \underbrace{C[(R-1)l+1], \ldots, C[Rl]}_{G_R}. \tag{7}$$

For the convenience of description, we define $G_m = G_m[1]G_m[2] \cdots G_m[l]$ and $G_m[i] = C[(m-1) * l + i]$, $m = 1, 2, \ldots, R$.

(3) *Processing*. The following operations are performed on the 2D CML, as illustrated by Fig. 5.

Step 1–Step R: Define $a \in [1, R]$ as the step index. In Step a, the lattice values in the second and fourth rows are modified according to Eq. (8):

$$x_{K(a-1)+1}^{2,j} = G_a[j], \quad x_{K(a-1)+1}^{4,j} = G_a[8 + j] \qquad (j = 1, 2, \ldots, 8). \tag{8}$$

Then according to Eq. (4), iterate the 2D CML for K times and change the system from State $K(a-1)+1$ to State $Ka+1$.

Step $(R+1)$–Step $2R$: Define $b \in [R+1, 2R]$ as the step index. In Step b, the lattice values in the second and fourth rows are modified according to Eq. (9):

$$x_{K(b-1)+1}^{2,j} = G_{2R-b+1}[j], \quad x_{K(b-1)+1}^{4,j} = G_{2R-b+1}[8 + j] \qquad (j = 1, 2, \ldots, 8). \tag{9}$$

Then according to Eq. (4), iterate the 2D CML for K times and change the system from State $K(b-1)+1$ to State $Kb+1$. The final state of 2D CML is $2KR+1$.

(4) Transform the final lattice values in the first and third rows of 2D CML, i.e., $x_{2KR+1}^{i,j}$ $(i = 1, 3; j = 1, 2, \ldots, 8)$, to the corresponding binary format and extract 8 bits (9th to 16th bits after the decimal point) from each x. Finally, juxtaposes these bits from left to right to get a 128-bit hash value.

3.3. Determining the minimum number of K

We determine the minimum number of iterations K in each step such that the distribution of each lattice value, even for a tiny change, is independent of the corresponding distribution of $x_{n+K}^{i,j}$ $(i = 1, \ldots, 4; j = 1, \ldots, 8)$ with χ^2 test as follows [4]:

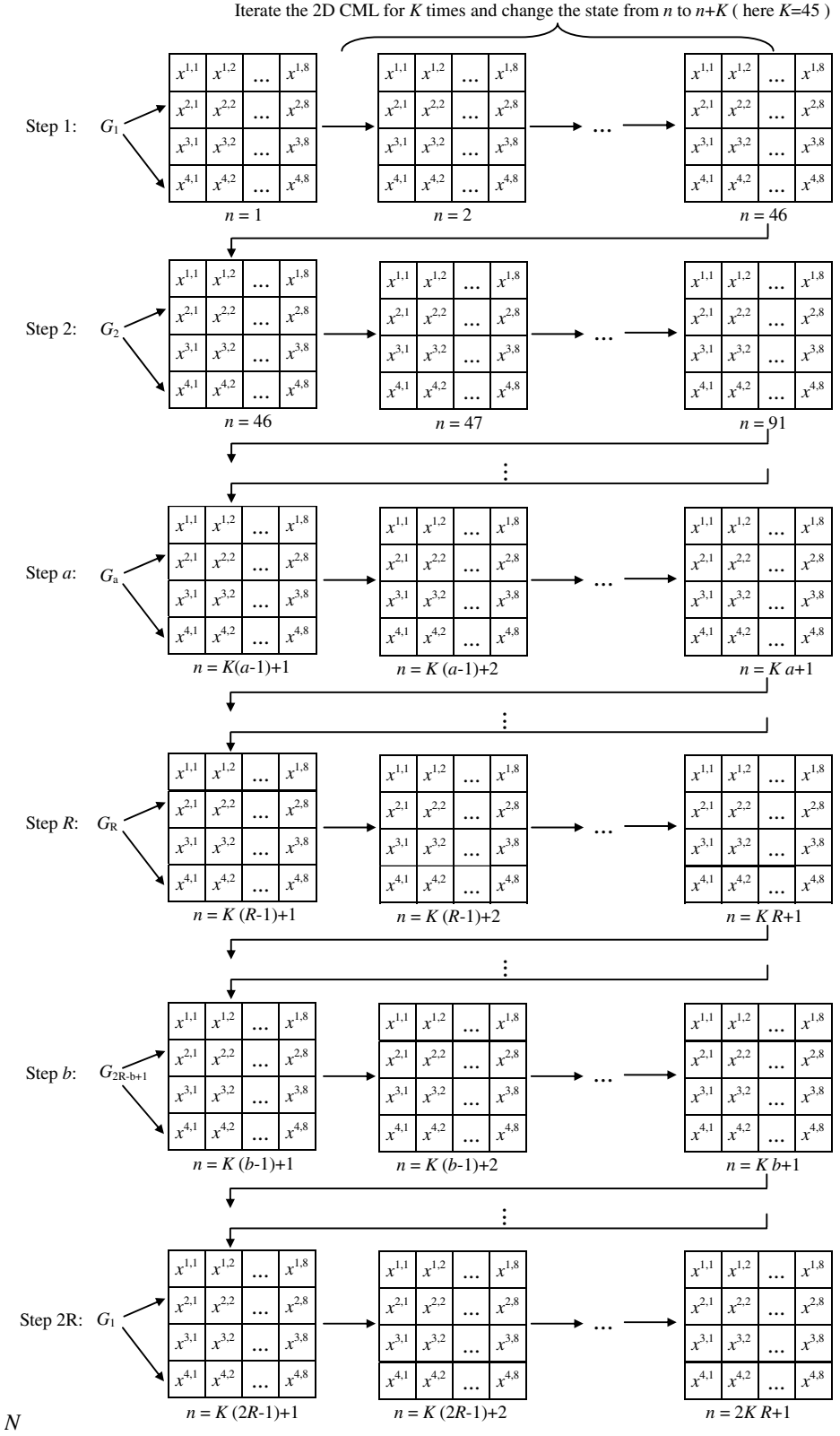

Fig. 5. Detail operations from Step 1 to Step 2*R*.

(1) Divide the interval $[0,1]$ into m small intervals $[(d/m),(d+1)/m]$, $d=0,1,\ldots,m-1$.
(2) Randomly set the initial values of the lattices in 2D CML and iterate the 2D CML for K times. We get $x_K^{i,j}$ $(i=1,\ldots,4;\ j=1,\ldots,8)$. Then change the initial state of 2D CML by adding $\Delta\alpha$ to a randomly chosen lattice. Iterate the 2D CML for K times again and obtain $(x')_K^{i,j}$ $(i=1,\ldots,4;\ j=1,\ldots,8)$. We compute N pairs of $x_K^{i,j}$ and $(x')_K^{i,j}$. For each lattice, make a $m\times m$ contingency table about frequency n_{ef} of $(x_K^{i,j},(x')_K^{i,j})$ satisfying $(e/m)<x_K^{i,j}<(e+1)/(m)$, $(f/m)<(x')_K^{i,j}<(f+1)/(m)$.
(3) For each lattice, calculate its χ^2 according to Eq. (10):

$$\chi^2=N\left(\sum_{e=1}^{m}\sum_{f=1}^{m}\frac{n_{\mathrm{ef}}}{\sum_{f=1}^{m}n_{\mathrm{ef}}\sum_{e=1}^{m}n_{\mathrm{ef}}}-1\right) \tag{10}$$

and choose the maximum χ^2 from all lattices as the result of χ^2-test. The result is shown in Fig. 6. In our scheme, $m=11$, $N=1000$ and $\Delta\alpha=1/256$.

If the χ^2 values are smaller than the upper 5% critical point of χ^2 of which the number of the degrees of freedom is $(m-1)(m-1)$, the independence is not rejected at level of significance 0.05. From Fig. 6, we can see that the independence is not rejected when $K\geqslant 43$ because the upper 5% critical point of χ^2_{100} is 124.3. Leaving an assurance margin, we choose $K=45$ as the number of iterations in each step.

4. Performance analysis

4.1. Hash result of messages

The proposed algorithm is used to perform hash simulation under the following five kinds of conditions:

Condition 1: The original message is "*Cryptographic hash functions play a fundamental role in modern cryptography. While related to conventional hash functions commonly used in non-cryptographic computer applications – in both cases, larger domains are mapped to smaller ranges – they differ in several important aspects. Our focus is restricted to cryptographic hash functions (hereafter, simply hash functions), and in particular to their use for data integrity and message authentication*".
Condition 2: Change the first character *C* in the original message to *D*.
Condition 3: Change the word *functions* in the original message to *function*.
Condition 4: Change the full stop at the end of the original message to a comma.
Condition 5: Add a blank space to the end of the original message.

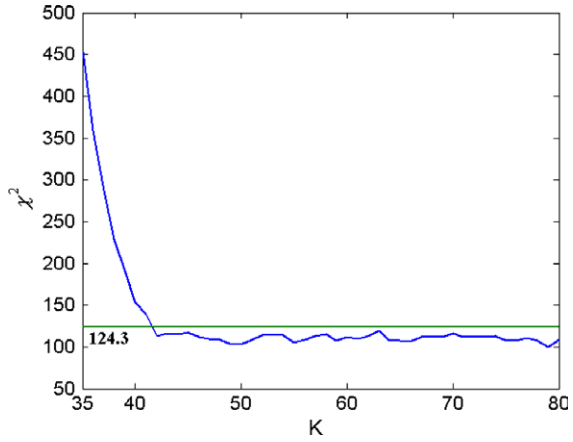

Fig. 6. Result of χ^2 test.

The corresponding hash values in hexadecimal format are given as follows, followed by the corresponding number of different bits compared with the hash value obtained under Condition 1:

Condition 1: 931786912B6E1E6B7E5454D86DFA6754.
Condition 2: ADA1B5465DFD356BCA88A0C9C555F52E (66).
Condition 3: 822783BDF27EA4A4908A1E3250455BAC (67).
Condition 4: C808EE4D57CBC747ADFDCD063218B87E (74).
Condition 5: 8388AE2B92D42D4F431C3E489012D449 (63).

The corresponding graphical display of binary sequences is shown in Fig. 7. The simulation results indicate that a tiny difference in the message causes huge changes in the final hash value.

4.2. Statistical analysis of diffusion and confusion

Confusion and diffusion are two basic design criteria for encryption algorithms, including hash functions. Diffusion means spreading out the influence of a single plaintext symbol over many ciphertext bits so as to hide the statistical structure of the plaintext. Confusion means the use of transformations to complicate the dependence of ciphertext statistics on plaintext statistics. For the hash value in binary format, each bit can be only 0 or 1. Therefore, the ideal diffusion effect should be that any tiny changes in the initial condition lead to a 50% changing probability of each bit. Usually, six statistics are defined as follows:

Minimum changed bit number: $B_{\min} = \min(\{B_i\}_1^N)$.
Maximum changed bit number: $B_{\max} = \max(\{B_i\}_1^N)$.
Mean changed bit number: $\overline{B} = \frac{1}{N}\sum_1^N B_i$.
Mean changed probability: $P = (\overline{B}/128) \times 100\%$.
Standard variance of the changed bit number: $\Delta B = \sqrt{\frac{1}{N-1}\sum_{i=1}^N (B_i - \overline{B})^2}$.
Standard variance: $\Delta P = \sqrt{\frac{1}{N-1}\sum_{i=1}^N (B_i/128 - P)^2} \times 100\%$.
where N is the total number of tests and B_i is the number of changed bits in the ith test.

We have performed the following diffusion and confusion test: A paragraph of message is randomly chosen and the corresponding hash value is generated. Then a bit in the message is randomly selected and toggled and a new hash value is generated. Finally, the two hash values are compared with each other.

This kind of test is performed N times, and the corresponding distribution of changed bit number is plotted in Fig. 8, where $N = 10{,}000$. Obviously, the changed bit number corresponding to 1 bit changed message concentrates around the ideal changed bit number, i.e., 64 bits. It indicates that the algorithm has very strong capability for diffusion and confusion.

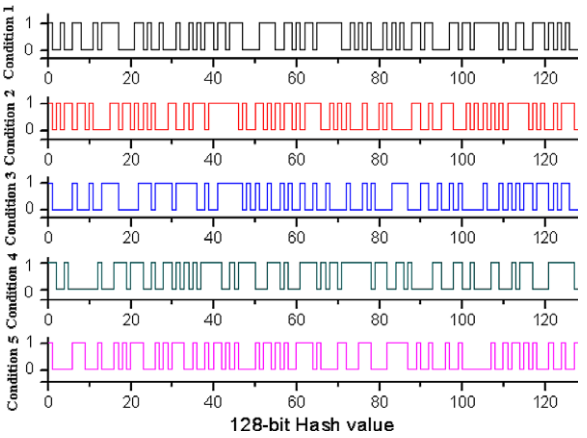

Fig. 7. Hash values under different conditions.

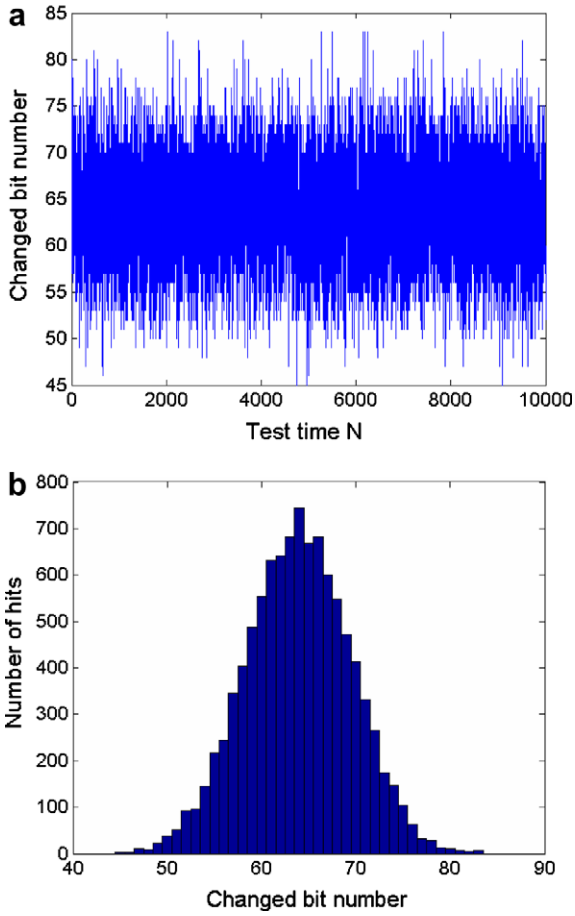

Fig. 8. Distribution of changed bit number: (a) plot of B_i; (b) histogram of B_i.

The test results with $N = 256$, 512, 1024, 2048, 10,000, respectively, are listed in Table 1. Based on the results, we can draw the following conclusion: the mean changed bit number $\overline{B}$ and the mean changed probability P are both very close to the ideal value 64 bits and 50%. While ΔB and ΔP are very small, which indicates the diffusion and confusion capability is very stable. Thus, a slight difference in the plaintext will cause great changes in the hash value, which contributes to the high plaintext-sensitivity of our hash function. This property is important in maintaining the secrecy against statistical attacks.

4.3. Collision analysis

4.3.1. Birthday-attack

Collision resistance and birthday-attack are related to each other. Both are originated from the probability problem that two random input data are found to hash to the same value. In the proposed algorithm,

Table 1
Statistical performance of our algorithm

	$N = 256$	$N = 512$	$N = 1024$	$N = 2048$	$N = 10{,}000$
$\overline{B}$	64.09	63.92	64.08	63.98	63.90
P (%)	50.07	49.94	50.06	49.98	49.91
ΔB	5.38	5.62	5.56	5.53	5.58
ΔP (%)	4.21	4.39	4.34	4.33	4.36
$B_{\min}$	50	44	47	47	45
$B_{\max}$	82	81	81	83	83

the state of 2D CML is related to each message bit. By iterations, substantial changes are obtained at the final state even if there is only a one-bit change in the message. According to the above analysis, our algorithm is secure against statistical attacks. For birthday-attack, the security of the cryptosystem is determined by the length of the hash value, which is 128-bit in our proposed function. According to the definition of birthday-attack [13,15,16], the attack difficulty is 2^{64}. Due to the advancement in computing power, the required hash length will continue to increase. By adjusting the size of 2D CML or extracting more bits from each lattice value, a longer hash value can be obtained. Therefore, our hash function can resist this kind of attack.

4.3.2. Meet-in-the-middle attack

The meet-in-the-middle attack seeks collisions on intermediate results (i.e., chaining variables) rather than the overall hash result [13,15,16]. In our scheme, the intermediate results are the floating-point values of the lattices. The iterations of 2D CML can be considered as the hash round function. In Section 3.3, we know that if there is only one bit difference (i.e., $\Delta\alpha = 1/256$) between two input blocks, the two lattice values are independent after 45 iterations. In the floating-point domain, it is infeasible to find two different input blocks, which have the same lattice values after 45 iterations. As a result, our hash function can resist meet-in-the middle attack.

4.3.3. Collision test

We have performed the following test for the quantitative analysis on collision resistance [24]: first, the hash value for a paragraph of randomly chosen message is generated and stored in ASCII format. Then a bit in the message is selected randomly and toggled. A new hash value is then generated. The two hash values are compared and the number of ASCII characters with the same value at the same location is counted. Moreover, the absolute difference between the two hash values is calculated by the following formula:

$$d = \sum_{i=1}^{N} |t(e_i) - t(e'_i)|, \tag{11}$$

where e_i and e'_i is the ith ASCII character of the original and the new hash value, respectively. The function $t(\,)$ converts the entries to their equivalent decimal values. This kind of collision test is performed 10,000 times. The maximum, mean, and minimum values of d are listed in Table 2. The distribution of the number of ASCII characters with the same value at the same location in the hash value is given in Fig. 9. Notice that the maximum number of equal characters is only 2 and the collision probability is very low.

4.4. Flexibility

Due to the advances in technology, NIST plans to replace SHA-1 by longer and stronger hash functions (SHA-224, SHA-256, SHA-384 and SHA-512) by 2010 [14], as long hash values are needed in the future. To produce a long hash value, we introduce two simple modifications in our algorithm: increase the size of 2D CML or extract more bits from each lattice value. For example, a 256-bit hash value is obtained by using a 2D CML model with 4×16 lattices and extracting 8 bits from each lattice value in the first and third rows, or

Table 2
Absolute difference of two hash values

Absolute difference	Maximum	Minimum	Mean
Xiao's scheme	2221	696	1506
Zhang's scheme [30]	2022	565	1257
MD5	2074	590	1304
Our scheme (128 bits)	2295	689	1526
SHA-1	2730	795	1603
Our scheme (160 bits)	2745	822	1763

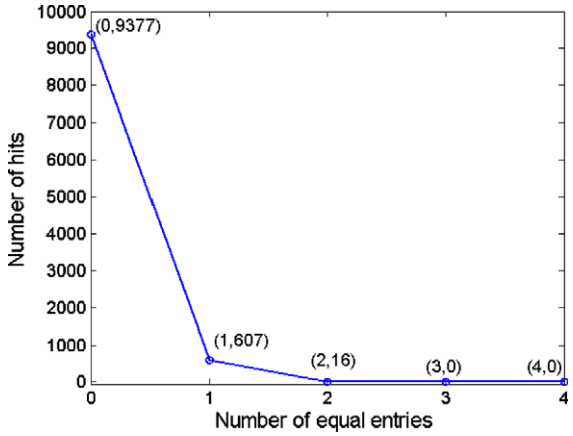

Fig. 9. Distribution of the number of ASCII characters with the same value at the same location in the hash value.

by extracting 16 bits from each lattice value in the origin 2D CML. Compared with the conventional hash algorithm such as MD5 with fixed 128-bit length, the proposed algorithm can adapt to the actual demand.

Although our proposed algorithm aims at unkeyed hash function, it can also be used to construct keyed hash function by simply treating the initial values as a secrete key. Since the secrete key consists of 16 characters, the key space is huge enough to resist exhaustive key search. Furthermore, the hash value is sensitive to the secrete key, too. Therefore, our algorithm can also be used for data origin authentication.

4.5. Comparison with other algorithms

4.5.1. Comparison with other algorithms based on chaos

To compare with other algorithms based on chaos, the corresponding data from [27,29,30] are tabulated in Tables 3–5. From Tables 1 and 3–5, it can be concluded that the statistical performance of these chaos-based algorithms approaches the ideal performance and satisfies the requirement of resisting statistical attacks. In particular, our scheme and the algorithm proposed in [30] have better statistical performance.

The absolute difference between two hash values in [27,30] are shown in Table 2. The maximum number of equal characters at the same location in the hash values generated using [27,30] are listed in Table 6.

Table 3
Statistical performance of algorithm in [27]

	$N = 256$	$N = 512$	$N = 1024$	$N = 2048$
$\overline{B}$	64.4414	63.8008	63.8398	63.8501
P (%)	50.34	49.84	49.87	49.88
ΔB	5.5218	5.7081	5.7078	5.7889
ΔP (%)	4.31	4.46	4.46	4.52

Table 4
Statistical performance of algorithm in [29]

	$N = 256$	$N = 512$	$N = 1024$	$N = 2048$
$\overline{B}$	63.35	64.62	63.41	64.43
P (%)	49.32	50.53	49.49	50.46
ΔB	5.934	5.816	5.675	5.568
ΔP (%)	5.013	4.927	4.684	4.506
$B_{\max}$	69	73	78	85
$B_{\min}$	51	48	47	44

Table 5
Statistical performance of algorithm in [30]

	$N = 512$	$N = 1024$	$N = 2048$	$N = 10{,}000$
$\overline{B}$	63.98	63.94	63.91	63.96
P (%)	49.98	49.95	49.92	49.97
ΔB	5.53	5.31	5.58	5.52
ΔP (%)	4.33	4.15	4.36	4.32
$B_{\max}$	81	81	80	85
$B_{\min}$	44	46	46	44

Table 6
Maximum number of equal characters at the same location in the hash value

	Zhang's scheme [30]	Xiao's scheme	MD5	SHA-1	Our scheme (128 bits)	Our scheme (160 bits)
Maximum number	2	3	2	2	2	2

Obviously, the bigger the absolute difference or the smaller the maximum number of equal characters is, the stronger the collision resistance is. Hence our algorithm possesses a stronger collision resistance than the algorithms studied in [27,30].

4.5.2. Comparison with MD5 and SHA-1

According to the approach described in Section 4.2, we perform tests for MD5 and SHA-1 using the same messages. The statistical results are given in Tables 7 and 8. Since SHA-1 produces a 160-bit hash value, we extend the message digest size of our algorithm from 128 bits to 160 bits by extracting 10 bits (9th to 18th bits after the decimal point) from the final lattice values in the first and third rows of the 2D CML. The statistical results of our modified algorithm are listed in Table 9. Meanwhile, according to the approach described in Section 4.3.3, we perform the quantitative test of collision for MD5, SHA-1 and our modified algorithm using the same messages. This kind of collision test is performed 10,000 times. The absolute difference between the two corresponding hash values and the maximum number of equal characters at the same location in the hash values are shown in Tables 2 and 6, respectively.

Table 7
Statistical performance of MD5

	$N = 256$	$N = 512$	$N = 1024$	$N = 2048$	$N = 10{,}000$
$\overline{B}$	63.68	63.92	63.98	64.03	64.05
P (%)	49.75	49.93	49.98	50.02	50.04
ΔB	5.38	5..78	5.73	5.66	5.68
ΔP (%)	4.20	4.36	4.48	4.42	4.44
$B_{\min}$	49	49	42	44	42
$B_{\max}$	78	82	81	86	83

Table 8
Statistical performance of SHA-1

	$N = 256$	$N = 512$	$N = 1024$	$N = 2048$	$N = 10{,}000$
$\overline{B}$	79.94	80.38	80.01	79.86	79.99
P (%)	49.96	50.23	50.01	49.91	49.99
ΔB	5.69	6.53	6.23	6.24	6.27
ΔP (%)	3.55	4.08	3.89	3.91	3.92
$B_{\min}$	65	58	60	58	58
$B_{\max}$	99	98	102	101	103

Table 9
Statistical performance of our modified algorithm

	$N = 256$	$N = 512$	$N = 1024$	$N = 2048$	$N = 10{,}000$
$\overline{B}$	79.98	80.37	80.26	80.01	80.03
P (%)	49.99	50.23	50.16	50.01	50.02
ΔB	5.97	6.46	6.23	6.34	6.31
ΔP (%)	3.73	4.04	3.89	3.97	3.94
$B_{\min}$	63	59	61	57	57
$B_{\max}$	99	98	101	102	103

Based on the result in Tables 1 and 7, the statistical performance of our algorithm is as good as that of MD5. Moreover, ΔB and ΔP of our algorithm are smaller than those of MD5, which indicate that our result is more stable. From Tables 8 and 9, we may conclude that our algorithm almost possesses the same statistical performance as SHA-1. According to the data in Tables 2 and 6, our algorithm has a bigger absolute difference between two hash values than MD5 and SHA-1. Meanwhile, the maximum numbers of equal character of MD5, SHA-1 and our algorithm are the same, i.e., only two, which indicates the collision chance is very low.

4.5.3. Speed analysis

Since our algorithm is based on a complex chaotic system, complicated computations are needed in producing a hash value and so our algorithm is slower than MD5, SHA-1 and algorithms in [27,30]. Nevertheless, it still possesses a sufficiently high hashing speed for practical use. The 2D CML model is a kind of spatiotemporal chaos. Compared with the algorithm in [29], which is also based on spatiotemporal chaos, our algorithm has a much higher efficiency. These two algorithms are implemented using Visual C++ running on a personal computer with a Pentium IV 2.4 GHz processor and 256 MB RAM. The execution time of the two algorithms with different message length is plotted in Figs. 10 and 11, respectively. As observed from these figures, the execution time of our algorithm is much shorter, especially when the message is long.

4.6. Further discussion

Although physical chaotic systems run over real number field with infinite word length, many researches show that digital realizations using floating-point arithmetic are close approximations of the continuous-value chaotic model and yield very good results if the fraction part is sufficiently long [30]. In our algorithm, the 2D CML is iterated with double precision floating-point arithmetic. Because IEEE 754 floating-point standard is available in virtually almost all computers produced since 1980 [3], two hash values of a message generated by

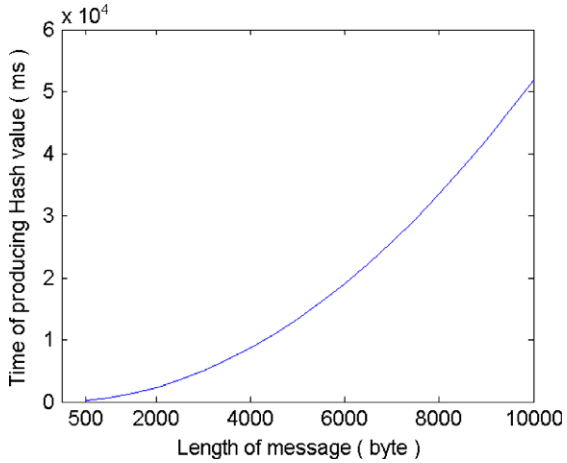

Fig. 10. Execution time of algorithm in [29].

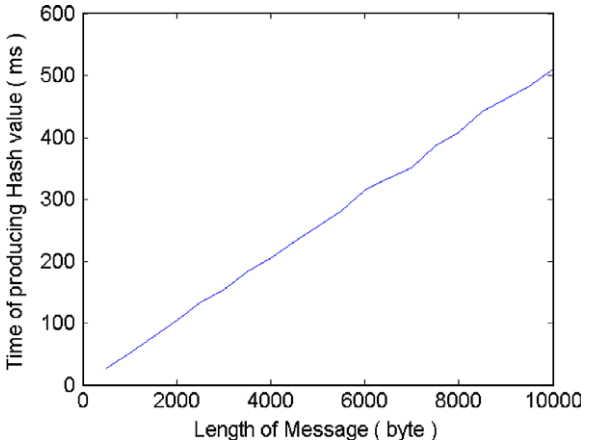

Fig. 11. Execution time of our algorithm.

two computers are identical if IEEE 754 floating-point standard is implemented in both computers, in spite of different operating systems, programming languages and word lengths.

5. Conclusion

In this paper, an algorithm for one-way hash function construction based on 2D CML is investigated. The parameters of the 2D CML are set based on their influence on the LLE so as to make the system more chaotic. The state of 2D CML is determined dynamically by the previous state and the corresponding message bit at different positions. The hash value is obtained through the linear transform on the final state of 2D CML. Since the 2D CML has multiple positive Lyapunov exponents, it is very difficult or even impossible to predict its time series. This leads to the good performance in diffusion and confusion. Substantial changes can be found in the final state of 2D CML even if only one bit is changed in the message. Theoretical analyses and numeric simulations have demonstrated that the proposed algorithm satisfy all the performance requirements of a strong hash function. It is practical and reliable, with high potential to be adopted in e-commerce applications.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable suggestions. The work described in this paper was supported by the National Natural Science Foundation of China (Nos. 60573047, 60703035), the Post-doctoral Science Foundation of China (No. 20060400714), the Foundation of Chongqing Education Committee (Nos. KJ070503, KJ070509), and the Natural Science Foundation of Chongqing University of Posts and Telecommunications (Nos. A2006-41, A2007-26).

References

[1] G. Alvarez, F. Montoya, M. Romera, G. Pastor, Cryptanalysis of dynamic look-up table based chaotic cryptosystems, Physics Letters A 326 (2004) 211–218.
[2] J.W. Byun, D.H. Lee, J.I. Lim, EC2C-PAKA: an efficient client-to-client password-authenticated key agreement, Information Sciences 177 (2007) 3995–4013.
[3] D. Goldberg, D. Priest, What every computer scientist should know about floating-point arithmetic, ACM Computing Surveys 23 (1) (1991) 5–48.
[4] T. Habutsu, Y. Nishio, I. Sasase, S. Mori, A secret key cryptosystem by iteration a chaotic map, in: Proceedings of Eurocrypt'91, Brighton, UK, 1991, pp. 127–140.
[5] F. Han, J. Hu, X. Yu, et al., Fingerprint images encryption via multi-scroll chaotic attractors, Applied Mathematics and Computation 185 (2007) 931–939.
[6] L. Kocarev, Chaos-based cryptography: a brief overview, IEEE Circuits and Systems Magazine 1 (3) (2001) 6–21.

[7] P. Li, Z. Li, W.A. Halang, G. Chen, A multiple pseudorandom-bit generator based on a spatiotemporal chaotic map, Physics Letters A 349 (2006) 467–473.
[8] S. Li, G. Chen, X. Mou, On the dynamical degradation of digital piecewise linear chaotic maps, International Journal of Bifurcation and Chaos 15 (10) (2005) 3119–3151.
[9] S. Lian, Z. Liu, Z. Ren, H. Wang, Hash function based on chaotic neural networks, in: Proceedings of the 2006 International Symposium on Circuits and Systems, 2006, pp. 237–240.
[10] W. Luo, Hashing via finite field, Information Sciences 176 (2006) 2553–2566.
[11] L.P. Maguire, B. Roche, T.M. McGinnity, L.J. McDaid, Predicting a chaotic time series using a fuzzy neural network, Information Sciences 112 (1998) 125–136.
[12] N. Masuda, K. Aihara, Cryptosystems with discretized chaotic maps, IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications 49 (1) (2002) 28–40.
[13] A. Menezes, P. van Oorschot, S. Vanstone, Handbook of applied cryptography, CRC Press, 1996.
[14] NIST Brief Comments on Recent Cryptanalytic Attacks on Secure Ha-shing Functions and the Continued Security Provided by SHA-1, 2004. http://csrc.nist.gov/hash_standards_comments.pdf.
[15] Secure Hash Standard, Federal Information Processing Standards Publication (FIPS PUB) 180-2, 2002.
[16] Security Requirements for Cryptographic Modules, Federal Information Processing Standards Publication (FIPS PUB) 140-1, 2002.
[17] K.M. Short, Unmasking a modulated chaotic communications scheme, International Journal of Bifurcation and Chaos 6 (2) (1996) 367–375.
[18] K.M. Short, Steps toward unmasking secure communications, International Journal of Bifurcation and Chaos 4 (4) (1994) 959–977.
[19] S. Wang, W. Liu, H. Lu, et al., Periodicity of chaotic trajectories in realizations of finite computer precisions and its implication in chaos communications, International Journal of Modern Physics B 18 (2005) 2617–2622.
[20] X. Wang, D. Feng, X. Lai, H. Yu, Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD, Rump Session of Crypto'04 E-print, 2004.
[21] X. Wang, X. Lai, D. Feng, et al., Cryptanalysis of the Hash functions MD4 and RIPEMD, in: Proceedings of Eurocrypt'05, Aarhus, Denmark, 2005, pp. 1–18.
[22] X. Wang, H. Yu, How to break MD5 and other hash functions, in: Proceedings of Eurocrypt'05, Aarhus, Denmark, 2005, pp. 19–35.
[23] Alan Wolf, Jack B. Swift, L. Harry, et al., Determining Lyapunov exponents from a time series, Physica D 16 (1985) 285–317.
[24] K. Wong, A combined chaotic cryptographic and hashing scheme, Physics Letters A 307 (2003) 292–298.
[25] T. Xiang, K. Wong, X. Liao, A novel symmetrical cryptosystem based on discretized two-dimensional chaotic map, Physics Letters A 364 (2007) 252–258.
[26] D. Xiao, X. Liao, S. Deng, A novel key agreement protocol based on chaotic maps, Information Sciences 177 (2007) 1136–1142.
[27] D. Xiao, X. Liao, S. Deng, One-way Hash function construction based on the chaotic map with changeable-parameter, Chaos Solitons & Fractals 24 (2005) 65–71.
[28] X. Yi, Hash function based on chaotic tent maps, IEEE Transactions on Circuits and Systems II 52 (6) (2005) 354–357.
[29] H. Zhang, X. Wang, Z. Li, D. Liu, One way Hash function construction based on spatiotemporal chaos, Acta Physica Sinica 54 (9) (2005) 4006–4011 (in Chinese).
[30] J. Zhang, X. Wang, W. Zhang, Chaotic keyed hash function based on feedforward–feedback nonlinear digital filter, Physics Letters A 362 (2007) 439–448.
[31] Y. Zhou, B. Fang, Z. Cao, X. Yun, X. Cheng, How to construct secure proxy cryptosystem, Information Sciences 177 (2007) 4095–4108.