One-way hash function construction based on the spatiotemporal chaotic system*

Luo Yu-Ling(罗玉玲)† and Du Ming-Hui(杜明辉)

School of Electronic and Information Engineering, South China University of Technology, Guangzhou 510641, *China*

(Received 15 November 2011; revised manuscript received 6 December 2011)

Based on the spatiotemporal chaotic system, a novel algorithm for constructing a one-way hash function is proposed and analysed. The message is divided into fixed length blocks. Each message block is processed by the hash compression function in parallel. The hash compression is constructed based on the spatiotemporal chaos. In each message block, the ASCII code and its position in the whole message block chain constitute the initial conditions and the key of the hash compression function. The final hash value is generated by further compressing the mixed result of all the hash compression values. Theoretic analyses and numerical simulations show that the proposed algorithm presents high sensitivity to the message and key, good statistical properties, and strong collision resistance.

Keywords: hash value, spatiotemporal chaos, hash compression function

PACS: 05.45.–a, 05.45.Gg, 05.45.Ra **DOI:** 10.1088/1674-1056/21/6/060503

1. Introduction

With the popularization of digital networks, especially the rapid development of electronic commerce and electronic government affairs, information security issues are attracting more and more attention. As a ubiquitous physical process in nature, chaos has been applied to the design of cryptographic systems, including image encryption,[1−5] synchronization communication,[6−8] and the hash function,[9−33] due to its analogous cryptography characteristics such as extremely sensitivity to initial conditions and system parameters, randomness and ergodicity. The cryptographic hash function is a key technology for information security and plays an important role in data integrity, message authentication and digital signatures,[9−11] which is a one-way cryptosystem used to compress an arbitrary length message into a fixed length value. Since Rivest constructed MD4 in 1990,[12] several conventional hash functions have been presented,[13] i.e., MD5, SHA-0, SHA-1, PIPEMD, SHA-2. Among them, MD5 and SHA-1 are the two most popular hash functions and usually used for all kinds of networks. However, these conventional hash functions are constructed by using either complicated methods based on arithmetical operations or multi-round iterations of some available ciphers.[14] Moreover, MD5, SHA-1, and PIPEMD were successfully attacked by Wang *et al.*,[15−17] and collision of SHA-0[18] can be found within the running-time of about 2^{40} SHA-0 algorithms. Meanwhile, such results have led to a lack of confidence in SHA-2, due to its similarity to MD5 and SHA-1.[19] So some new algorithms for constructing an efficient and secure hash function have become a key point for researchers.

Recently, some hash function constructions based on simple chaotic maps or chaotic neural networks were successively presented, each of which has a higher security level. For example, Wang *et al.*[20] introduced a hash function based on extended chaotic map switch in 2003. Peng *et al.*[21] and Zhang *et al.*[22] presented a hash function based on two-dimensional hyper-chaotic mappings and the spatiotemporal chaos in 2005, respectively. The above three algorithms are insecure, and Wang *et al.*[23] pointed out the potential collision flaws of this kind of method for constructing one-way hash function in Refs. [20]–[22]. In 2008, Zheng *et al.*[24] constructed the generalized Henon-based hash function; Yang and Gao[25] proposed a one-way hash function based on a hyper-chaotic cellular neural network; Long *et al.*[26] structured a one-way hash function based on an unified chaotic system. Although

*Project supported by the National Natural Science Foundation of China (Grant No. U0735004) and the Natural Science Foundation of Guangdong Province, China (Grant No. 05006593).

†Corresponding author. E-mail: luo.yuling@mail.scut.edu.cn

© 2012 Chinese Physical Society and IOP Publishing Ltd

these algorithms improve security to some extent, the efficiency of their practical application is not high. Then, two algorithms for keyed hash function construction based on chaotic neural network and spatiotemporal chaos were first proposed in 2009.[27,28] Soon afterwards, Wang and Zhao[29] analysed the weak keys and forgery attacks against the algorithm in Ref. [27] and took some measures to improve its security, and Wang and Shan[30] indicated the collisions existing in Ref. [28] and proposed an improved algorithm to avoid them. On that basis, some new parallel hash functions based on a spatiotemporal chaotic system or a chaotic neural network are presented.[31−33]

Compared with simple chaotic maps, spatiotemporal chaos greatly enhances the spatiotemporal complexity of nonlinear dynamics and mixtures, which can effectively prevent a raider from breaking the hash function. Therefore, in this article, we utilize the spatiotemporal chaotic map to construct a one-way hash function. Both the information about each message block and its position in the whole message block chain constitute the initial conditions and key of the spatiotemporal chaotic system, namely, the hash compression function. The final hash value is generated by further compressing the mixed result of all the hash compression values. Theoretic analyses and numerical simulation show that the proposed algorithm possesses high message and key sensitivity, good statistical properties, and strong collision resistance.

2. Preliminaries

2.1. Spatiotemporal chaos

In this paper, a spatiotemporal chaotic system is taken as the mold of the compression function h, which is a dynamical system with discrete-time, discrete-space, and continuous states. It is composed of nonlinear lattice maps, and each local map is coupled with other local maps according to a certain coupling rule. The spatiotemporal chaos system is defined as follows:[34]

$$x_{n+1}(i) = (1-\varepsilon)f(x_n(i)) + 0.5\varepsilon[f(x_n(i-1)) + f(x_n(i+1))], \quad (1)$$

where n is the time index, i $(i = 1, 2, \ldots, L)$ is the lattice site index, $\varepsilon \in (0, 1)$ is the coupling constant, and $x_n(i)$ represents the state variable for the i-th site at time n. The periodic boundary condition $x_n(i) = x_n(i+L)$ is assumed, and here the local mapping function f is described as follows:

$$f(x_n(i)) = \mu x_n(i)(1 - x_n(i)). \quad (2)$$

When the parameter $\mu \in (3.5699456\,4]$, the system is in a chaotic state as demonstrated in Fig. 1.

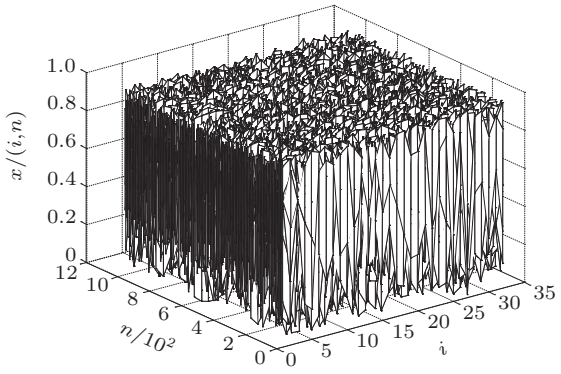


Fig. 1. Chaotic behaviour of system (1).

2.2. Prime module congruence method (PMCM)

PMCM is a common pseudo-random number generator (PRNG). It is described as follows:

$$\begin{cases} x_n = a x_{n-1} \bmod M, \\ r_n = x_n / M, \end{cases} \quad (3)$$

where $M \in Z^+$ is the modulus and $a \in Z^+$ is the multiplication factor, $x_n \in [0, M]$ is the generated pseudo random sequence, and $r_n \in (0, 1)$ is the corresponding random decimal. In general, there are two specific types: $a = 3125$, $M = 2^{35} - 31$, and $a = 16807$, $M = 2^{31} - 1$.

2.3. One-way hash function

The hash value is generated by the function form,

$$H = H(M) = \sum_i h(M_i), \quad (4)$$

where h is the compression function, M_i is the corresponding group message and $\sum$ denotes a nonlinear combination. A secure one-way hash function must satisfy the following properties.[35]

1) The function $H(\ )$ is keyed one-way, i.e., (i) given the key and M, it is easy to compute $H(M)$; (ii) without the key information, it is hard to find M when $H(M)$ is given; (iii) without the key information, it is hard to find $H(M)$ when M is given.

2) The function $H(\ )$ has sufficient confusion and diffusion. It should uniformly distribute the message

digest in the message digest space. This thwarts statistical attacks.

3) The function $H(\ )$ is keyed collision free. That is, without the key information, it is difficult to find two diverse message M and M' such that $H(M) = H(M')$.

4) The function $H(\ )$ should produce a message digest with at least 128-bit to thwart the birthday attack.

5) The function $H(\ )$ should have enough key space to thwart exhaustive key searching.

Compared with simple chaotic mappings, in addition to good chaotic inherent characters, the spatiotemporal chaotic system has strong spatiotemporal complexity and mixture, which can form the solid theoretical foundation for excellent hash function construction.

3. Parallel hash algorithm construction

3.1. The whole algorithm structure

In the process of generating the hash value, a message of arbitrary length is first divided into blocks whose lengths each are determined by the compression function h. It is padded to make its length a multiple of the message block length. Then, these blocks are synchronously processed. The final hash value is a mixed result of output values from all compression functions. The whole structure of the hash function construction is painted and shown in Fig. 2.

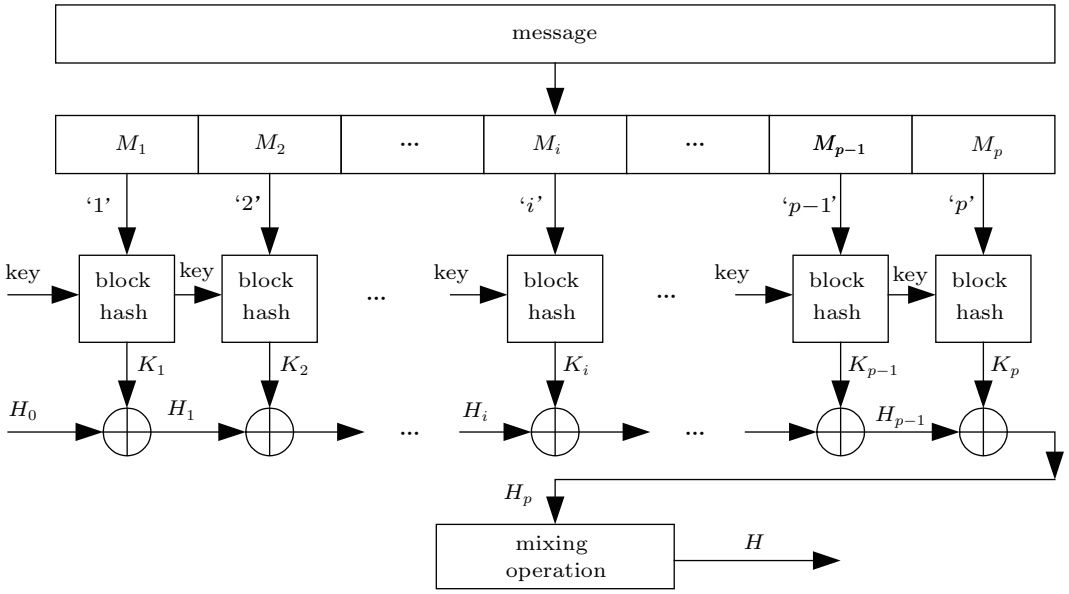


Fig. 2. The whole structure of hash function.

3.2. Algorithm description

Let $N \geq 128$ be the bit-length of the final hash value. In order to describe the algorithm in detail, here $N = 128$ is assumed. The concrete process of hash function construction is as follows:

Step 1 Let $l = 512$ be the bit-length of each message block. The original message M is padded such that its length is a multiple of l: let m be the length of the original message M, the padding bits $(100\ldots0)_2$ with length n (such that $(m+n) \bmod 512 = 512 - 64 = 448$, $1 \leq n \leq 512$) are appended. The left 64-bit denotes the length of the original message M. If m is greater than 2^{64}, then set $m = m \bmod 2^{64}$. So $M = (M_1, M_2, \ldots, M_p)$ is divided into p blocks with 512-bit, namely, 64 bytes.

Step 2 Set a 256-bit initial value H_0. Utilize PMCM mentioned above to generate a pseudorandom sequence with 64 numbers. Map the 64 decimals to a 256-bit binary value in terms of the rule in Table 1.

Step 3 According to the transform of the compression function h, generate the intermediate values $K_1, K_2, \ldots, K_p$ in parallel.

Step 4 Execute the operation according to Eq. (5),

$$\begin{cases} K_i = \text{block hash (key, } i\text{, } M_i), \\ H_p = H_0 \oplus K_1 \oplus K_2 \oplus \cdots \oplus K_i \cdots \oplus K_p. \end{cases} \tag{5}$$

Step 5 Further diffusion H_p. Divide the 256-bit binary number H_p into 32 binary numbers from left to right, i.e. every 8-bit forms a new binary number, and transform them into corresponding integers IN_j $(j = 1, 2, \ldots, 32)$. Then these integers IN_j are regarded as a single message block, and this new message block is implemented by the operations of the hash compression function. The final hash value H will be obtained from the compression value. Here, we point out that the lattice period is $L = 32$ now, and the coupling constant is $\varepsilon = 0.1$. Other operations are the same as those of the i-th message block in the above two steps.

The spatiotemporal chaotic system in Subsection 2.1 is used to generate the hash compression value. Because each message block has 64 characters with 8-bit, the lattice period is $L = 64$. The detailed operations to do the i-th message block M_i $(i = 1, 2, \ldots, p)$ are described as follows.

i) The 64-character ASCII IN_i^j of M_i should be pre-mapped into decimals d_i^j $(j = 1, 2, \ldots, 64)$ in (0,1) by means of linear transform $d_i^j = (IN_i^j + 0.8)/256$. And they are set as the initial conditions $x_0(j)$.

ii) The parameter μ of the local mapping function f and the coupling constant ε are supported by 64-character ASCII IN_i^j of the message block M_i and its position index i $(i = 1, 2, \ldots, p)$. They are $\mu = 3.9999995 - \text{round}(\bar{M}_i) \times 0.000001$ and $\varepsilon = (0.2 + i/2^{55})/2$, respectively, where $\bar{M}_i$ is the mean value of 64-character ASCII IN_i^j of M_i.

iii) After τ iterations of the spatiotemporal chaotic system under the initial conditions and parameters mentioned above, the orbit state results $x_\tau(j)$ $(j = 1, 2, \ldots, 64)$ are obtained, where $\tau = \text{round}\,(\bar{M}_i)$.

iv) The interval $(0, 1)$ is divided into 16 small intervals $[d/16, (d+1)/16)$, $d = 0, 1, \ldots, 15$. The initial one-to-one relationship between the 16 small intervals and the binary sequence is shown in Table 1.

v) The binary sequence in Table 1 is linked in series to assemble a 64-bit string, and the 64-bit string is done to cycle shift k times ($k = \tau \bmod 16$) in order to gain a new binary sequence with 16 elements. The result is demonstrated in Fig. 3.

vi) According to the one-to-one relationship between 16 small intervals and the new binary sequence, the 64-state results $x_\tau(j)$ are mapped to a 256-bit binary number K_i $(i = 1, 2, \ldots, p)$.

Table 1. Corresponding relations between decimal number and binary sequence.

Decimal number	Binary sequence	Decimal number	Binary sequence	Decimal number	Binary sequence	Decimal number	Binary sequence
(0,1/16)	0000 (0)	[1/4,5/16)	0100 (4)	[1/2,9/16)	1000 (8)	[3/4,13/16)	1100 (C)
[1/16,1/8)	0001 (1)	[5/16,3/8)	0101 (5)	[9/16,5/8)	1001 (9)	[13/16,7/8)	1101 (D)
[1/8,3/16)	0010 (2)	[3/8,7/16)	0110 (6)	[5/8,11/16)	1010 (A)	[7/8,15/16)	1110 (E)
[3/16,1/4)	0011 (3)	[7/16,1/2)	0111 (7)	[11/16,3/4)	1011 (B)	[15/16,1)	1111 (F)

0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

(a) The original binary sequence

0000	1001	0001	1010	0010	1011	0011	1100	0100	1101	0101	1110	0110	1111	0111	1000

(b) New binary sequence after the cycle shift for k times ($k = 3$)

Fig. 3. Specific process for cycle shift of the binary sequence. (a) The original binary sequence. (b) New binary sequence after the cycle shift for k times ($k = 3$).

It can be seen that the hash compression value K_i $(i = 1, 2, \ldots, p)$ corresponding to message block M_i $(i = 1, 2, \ldots, p)$ is generated in parallel, and it must be related to the corresponding key, the order and the content of current message block M_i, which can guarantee the security of the algorithm.

4. Security and performance analyses

4.1. Statistical distribution of hash value

The uniform distribution of hash values is one of the most important requirements for hash function

construction, which is directly related to the security of hash function. A simulation experiment has been done on the following paragraph of message: "Guangzhou is located in the middle south of Guangdong Province, north of the Pearl River Delta. It lies close to the South China Sea, Hong Kong, and Macau. Zhujiang (named also The Pearl River), the third largest river of China, runs through Guangzhou and is navigable to the South China Sea. Situated in such an excellent geographical region, Guangzhou is called China's South Gate. Covering an area of 7434.4 square kilometers, Guangzhou is home to more than 11 million people, including a 3.7 million."

Two-dimensional graphs are plotted to demonstrate the differences between the original message and the final hash value. In Fig. 4(a), the ASCII codes of the original message are localized within a small area, while in Fig. 4(b), the hexadecimal hash value is spread around very uniformly. A similar experiment is done to a paragraph of all "0" message with the same length as that of the original message. The contrast between message and the final hash value is also demonstrated in Fig. 5. Even under this very extreme condition, the difference is still distinct, and the distribution of the hash value is uniform. The simulation results indicate that there is a high influence of the original message on confusion and diffusion in the algorithm.

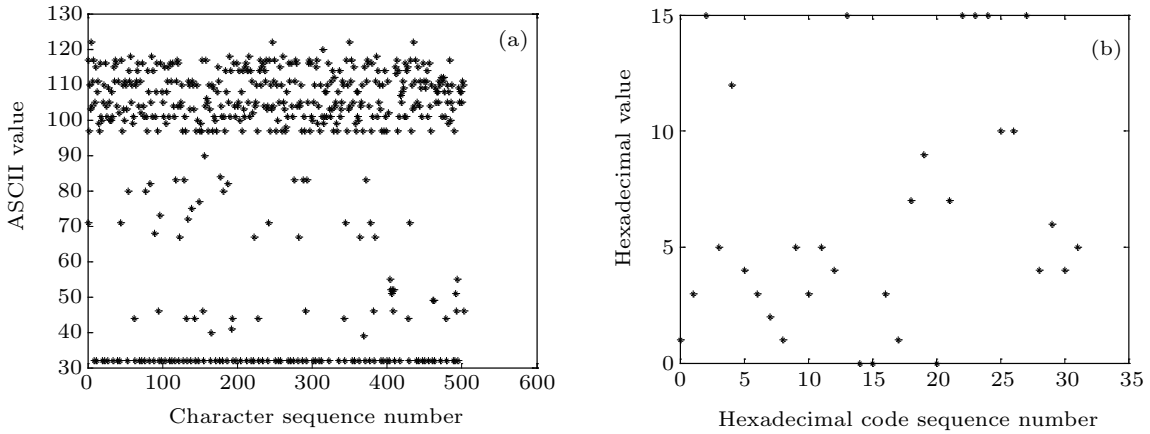


Fig. 4. Spread of hash values: (a) distribution of the original message in ASCII; (b) distribution of the hash value in hexadecimal format.

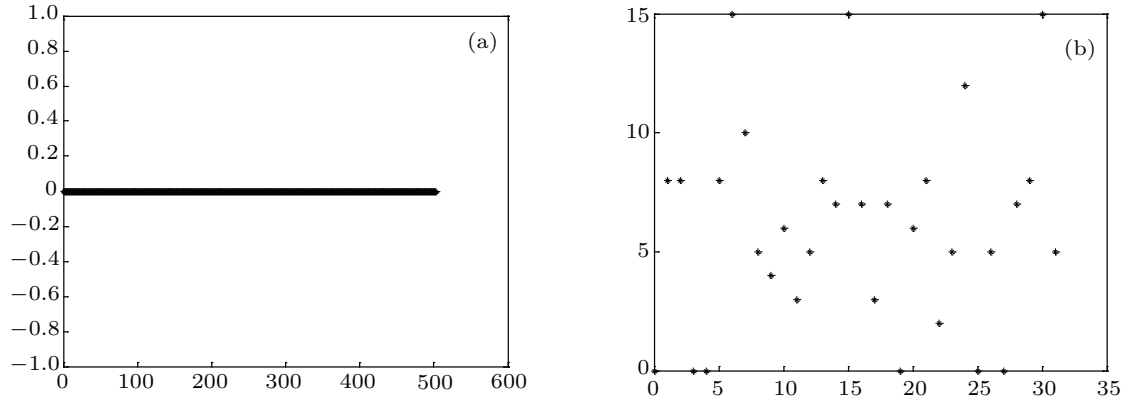


Fig. 5. Spread of hash values: (a) distribution of all '0'-message in ASCII; (b) distribution of the hash value in hexadecimal format.

4.2. Sensitivity of hash value to message

In order to evaluate the sensitivity of hash value to message, several simulation experiments are done under the following different six conditions:

C1: the original message mentioned above;

C2: change the first character G in the original message into H;

C3: change the number 11 in the original message into 10;

C4: change the full stop at the end of the original message into comma;

C5: change the word including in the original message into the word excluding;

C6: adds a blank space to the end of the original

message.

The corresponding hash values in hexadecimal format are gained as follows:

C1: 13F5C43215354F00317907FFFAAF4645;
C2: 003067CF1701690A0AF0E060A3407A47;
C3: 1B70505299AD3DEADBCCD72090B088D2;
C4: ACF99CB9BA7D82C7CCB9BABD47CF1E9B;
C5: 9CA4660F48F0C648B521294EEB34E694;
C6: F449B304350A095BC2713070240CEC9A.

The graphical display of binary sequence is shown in Fig. 6. The simulation result indicates that a tiny difference in the original message can cause a huge change in the final hash value.

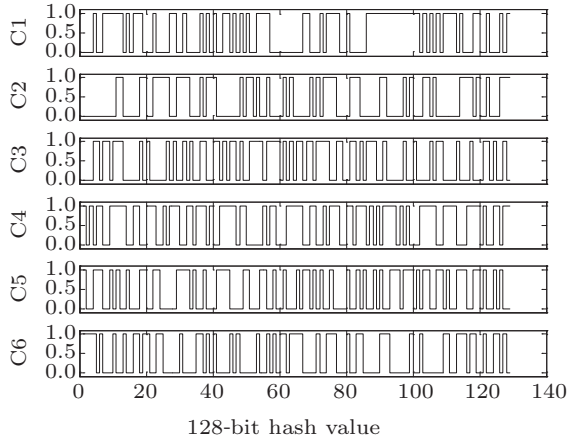


Fig. 6. Hash values of messages with the character difference.

4.3. Statistic analyses of confusion and diffusion

According to Shannon's information theory, diffusion and confusion are two basic principles to guide the design of an encryption algorithm, including hash function construction.[36] Hash function not only requires the effect of a single bit in original message that can spread around the whole hash value, but also hopes a complex relationship between the original message and the hash value that can guarantee the high probability property of the original message hidden. In the proposed algorithm, the original message information is mapped into parameter space, and the number of iterations for each message block is related to the corresponding message block. As is well known, each bit of the hash value in binary format is only 0 or 1, so the ideal diffusion effect should be that any tiny change in the original message under the same conditions leads to a 50% changing probability for each bit of the hash value. Here, four statistics are defined to test the confusion and diffusion of the hash function. They are as follows:

mean changed bit number: $\bar{B} = \frac{1}{N}\sum_{i=1}^{N} B_i$;

mean changed probability: $P = \left(\bar{B}/128\right) \times 100\%$;

mean-square deviation of the changed bit number:

$$\Delta B = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(B_i - \bar{B}\right)^2};$$

mean-square deviation of the changed probability:

$$\Delta P = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(B_i/128 - P\right)^2} \times 100\%,$$

where N is the total number of tests and B_i is the changed bit number in the i-th test. The message mentioned above is chosen to perform the test 256, 512, 1024, 2048, 3418 times, respectively. The experimental method is that a paragraph of message is randomly chosen, and the corresponding hash value is generated. Then a bit of the chosen message is randomly selected and toggled, and a new corresponding hash value is gained. Comparing two hash values, the number of changed bits is counted as B_i. The experimental results are shown in Fig. 7 and in Table 2, respectively.

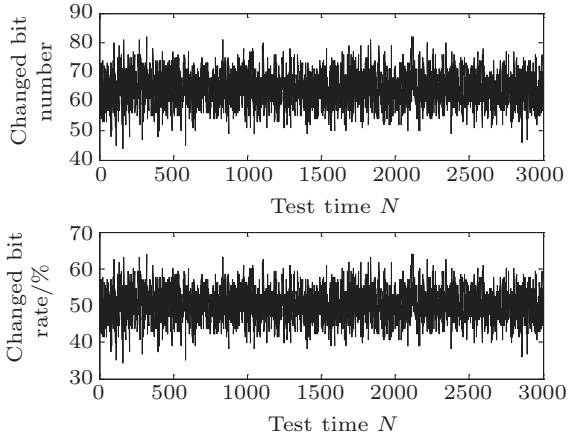


Fig. 7. Distributions of changed bit number and probability.

Table 2. Statistics of the number of bit changed B_i.

	$N = 256$	$N = 512$	$N = 1024$	$N = 2048$	$N = 3418$	Mean
$\bar{B}$	63.922	64.371	64.394	64.172	64.179	64.2706
ΔB	6.2774	6.2773	5.9459	5.7768	5.7498	6.00544
$P(\%)$	49.939	50.290	50.307	50.135	50.140	50.1622
$\Delta P(\%)$	4.9042	4.9041	4.6452	4.5132	4.492	4.69174

From the experiment results, we can see that both the mean changed bit number $\bar{B}$ and mean changed probability P are very close to their ideal values, while ΔB and ΔP are very small, which indicates the capability for confusion and diffusion is very stable. Therefore, the proposed algorithm for hash function construction approaches the ideal performance and satisfies the requirement for resisting statistical attacks.

4.4. Key security

The key security includes two aspects: the initial hash value H_0 with 256-bit and parameters of hash compression function in each message block. We perform the corresponding experiments in these two aspects, respectively. For each bit change of the initial hash value H_0, the distributions of changed bit number and probability of the final hash value are plotted in Fig. 8, and the corresponding four statistics are listed in Table 3. It can be seen that the influence of a single bit at any location of the initial hash value H_0 can spread around the whole hash value.

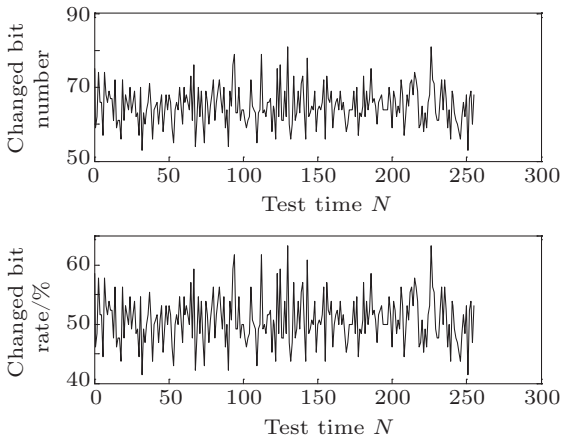


Fig. 8. Distributions of changed bit number and probability in the initial hash value H_0.

Table 3. Statistics of the number of bit changed B_i.

	$N = 50$	$N = 100$	$N = 150$	$N = 200$	$N = 256$	Mean
$\bar{B}$	64.600	64.980	64.907	64.990	64.996	64.8946
ΔB	5.0061	5.2531	5.5094	5.1529	5.2097	5.22624
$P(\%)$	50.469	50.766	50.708	50.773	50.778	50.6988
$\Delta P(\%)$	3.9110	4.1040	4.3042	4.0257	4.0701	4.083

On the other hand, the experimental result of the sensitivity of hash value to the parameters is shown in Fig. 9. The experimental conditions are set as follows.

C1: the original message is the same as the one in Subsection 3.1, and the original parameters are $\mu = 3.9999995 - \text{round}\,(\bar{M}_i) \times 0.000001$ and $\varepsilon = (0.2 + i/2^{55})/2$;

C2: the parameter μ is changed from $3.9999995 - \text{round}\,(\bar{M}_i) \times 0.000001$ to $3.999999500000001 - \text{round}\,(\bar{M}_i) \times 0.000001$;

C3: the parameter ε is changed from $(0.2 + i/2^{55})/2$ to $(0.200000000000001 + i/2^{55})/2$.

The corresponding hash values in hexadecimal format are obtained as follows:

C1: 13F5C43215354F00317907FFFAAF4645;

C2: 5EF1A20E03A9DE0F27A0FB4059C04090;

C3: 202404ED0993B21252C34924F4221247.

The graphical displays of binary sequences are shown in Fig. 9, indicating that the hash value is sensitive to the parameters within an accuracy of 10^{15} so that there is a large enough key space to resist the brute-force attack.

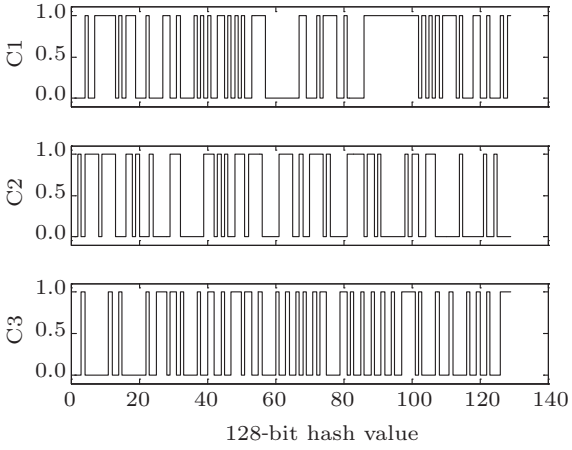


Fig. 9. Hash values of messages with the parameter difference.

4.5. Analysis of collision resistance

Two kinds of test methods are performed to conduct quantitative analysis on collision resistance. The first test method is as follows.[24−33] First, the hash value for a paragraph of message randomly chosen 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 and stored in ASCII for-

mat. Two hash values are compared, and the number of ASCII characters with the same value at the same location in the hash value, namely the number of hits, is counted. This kind of collision test is performed 2048 times, and the experimental result is given in Fig. 10. It is noticed that the maximal number of equal entries is only two and the collision is very low.

Moreover, the absolute difference between two hash values is calculated using the following formula:

$$d = \sum_{i=1}^{N} |e_i - e_i'|, \tag{6}$$

where e_i and e_i' denote the i-th decimal value of the corresponding ASCII character of the original hash value and the new hash value, respectively. The maximal, minimal, and mean values of d are listed in Table 4.

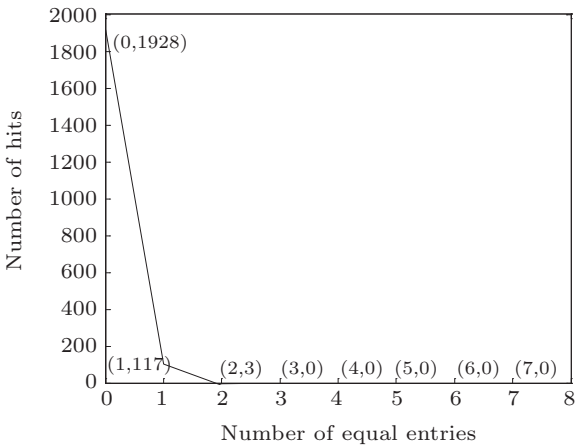


Fig. 10. Distribution of the number of ASCII characters with the same value at the same location in the hash value.

Table 4. Absolute difference between two hash values.

Absolute difference d	Maximal	Minimal	Mean
Our algorithm	2418	796	1598.6
Ref. [24]	2524	872	1675
Ref. [25]	2343	256	1314
Ref. [27]	1952	605	1227.8
Ref. [28]	2455	599	1439
Ref. [31]	1882	650	1251.2
Ref. [32]	2220	687	1432.1
Ref. [33]	2064	655	1367

The other test method is as follows.[20−23] First, the original message is taken as a byte, namely 8 bits, so the corresponding ASCII codes are from 0 to 255. Then the hash value is also taken as 8 bits, and the range of decimal hash value is the same as that of the original message, which guarantees that the hash value space is equivalent to the message space. Moreover, k denotes the number of a certain hash value mapping into the message space, and $n(k)$ denotes the number of k ASCII values in the hash value space. If $n(1)$ is greater, and others are smaller, the confusion and diffusion of hash function algorithm will be better and probability of collision will be less. Moreover, the collision resistance property of the hash function construction algorithm can be calculated using the formula: $R(k) = n(k)/\sum n(k)$. From the distribution of $n(k) - k$ in Fig. 11(a), we can obtain the distribution of $R(k) - k$ shown in Fig. 11(b), which indicates the good collision resistance property of the proposed hash function construction algorithm.

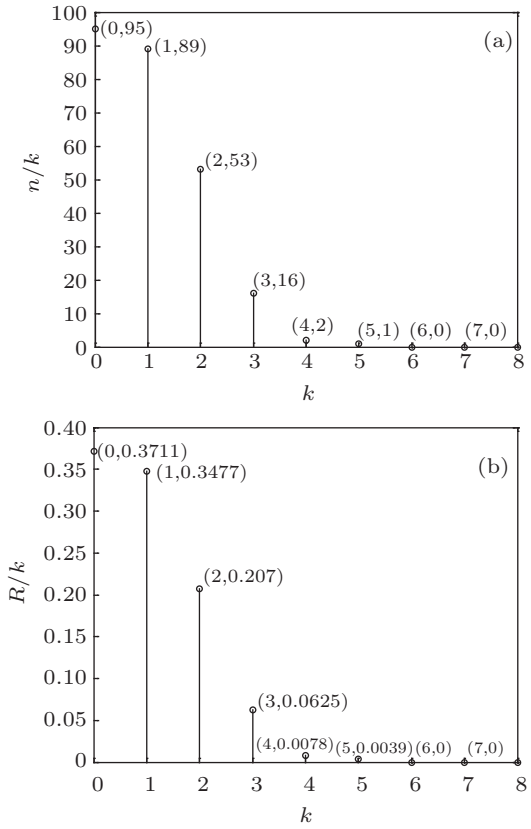


Fig. 11. Distribution of the $n(k) - k$ and the $R(k) - k$.

4.6. Comparison with other algorithms

Firstly, in order to compare the statistic performance with other algorithms, the corresponding data from Refs. [24]–[28] and [31]–[33] are listed in Table 5. For convenience of comparison, the statistic results of our algorithm are also given in Table 5. These results show that the statistic performances of all these functions are close to the ideal situation and so can resist statistical attacks. Among these algorithms, our scheme presents the best statistic performance.

In addition, the absolute difference between two hash values using various algorithms is obtained and

listed in Table 4. The maximal number of equal characters at the same location in the hash values is listed in Table 6. Obviously, the bigger the absolute difference or the smaller the maximal number of equal characters, the stronger the collision resistance is. According to the data listed in Table 4, our algorithm has a larger absolute difference between two hash values than other algorithms proposed. And it can be seen from Table 6 that the maximal number of equal characters among all algorithms, except for that cited from Ref. [25], is equal to 2. So our algorithm possesses a stronger collision resistance to some extent.

Table 5. Comparison of the statistic performances.

		$N = 128$	$N = 256$	$N = 512$	$N = 1024$	$N = 2048$	$N = 3418$	Mean
$\bar{B}$	Our algorithm	—	63.922	64.371	64.394	64.172	64.179	64.2706
	Ref. [25]	62.27	63.91	63.20	63.53	—	—	63.2275
	Ref. [27]	—	63.9453	64.1152	64.2666	64.0098	—	64.0842
	Ref. [28]	—	63.898	63.814	64.018	63.922		63.913
	Ref. [31]	—	63.5898	63.5586	64.1465	63.8804	—	63.7938
	Ref. [32]	—	63.7773	63.5605	63.5439	63.8076	—	63.6723
	Ref. [33]	—	63.893	63.900	64.009	64.153	—	63.986
$P(\%)$	Our algorithm	—	49.939	50.290	50.307	50.135	50.140	50.1622
	Ref. [25]	48.65	49.93	49.37	49.66	—	—	49.4025
	Ref. [27]	—	49.96	50.09	50.21	50.01	—	50.0675
	Ref. [28]	—	49.921	49.855	50.014	49.938	—	49.932
	Ref. [31]	—	49.86	49.66	50.11	49.91	—	49.685
	Ref. [32]	—	49.8260	49.6567	49.6437	49.8497	—	49.7440
	Ref. [33]	—	49.917	49.922	50.007	50.119	—	49.988
ΔB	Our algorithm	—	6.2774	6.2773	5.9459	5.7768	5.7498	6.00544
	Ref. [25]	0.59	0.42	0.30	0.21	—	—	0.38
	Ref. [27]	—	5.4012	5.5437	5.7882	5.7236	—	5.6142
	Ref. [28]	—	5.502	5.639	5.588	5.782	—	5.62775
	Ref. [31]	—	5.4557	5.5848	5.5622	5.7554	—	5.5895
	Ref. [32]	—	5.3915	5.6264	5.7143	5.7563	—	5.6221
	Ref. [33]	—	5.367	5.491	5.671	5.767	—	5.616
$\Delta P(\%)$	Our algorithm	—	4.9042	4.9041	4.6452	4.5132	4.492	4.69174
	Ref. [25]	4.33	3.13	2.19	1.54	—	—	2.7975
	Ref. [27]	—	4.22	4.33	4.52	4.47	—	4.385
	Ref. [28]	—	4.298	4.405	4.365	4.517	—	4.39625
	Ref. [31]	—	4.26	4.36	4.35	4.50	—	4.3675
	Ref. [32]	—	4.2121	4.3956	4.4643	4.4971	—	4.3923
	Ref. [33]	—	4.193	4.289	4.430	4.506	—	4.388

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

	Our algorithm	Ref. [24]	Ref. [25]	Ref. [27]	Ref. [28]	Ref. [31]	Ref. [32]	Ref. [33]
Maximal number	2	2	—	2	2	2	2	2

In addition, comparing the proposed algorithm with those cited from Refs. [28] and [30], there are some differences in several aspects as shown below.

(I) For the overall structure, the proposed algorithm in this paper is achieved in the parallel mode, that is to say, each message block is processed in parallel, while those of the algorithms in the Refs. [28] and [30] are realized in a sequential mode, i.e., the output state values of each message block are taken as the initial input values of the next one.

(II) The spatiotemporal chaos is chosen as the corresponding hash compression function in three papers, but the basic modules constituting the spatiotemporal chaotic system is different from each other. In this paper, the coupled map lattice (CML) is employed as the model to generate the corresponding spatiotemporal chaotic system and the logistic defined in Eq. (2) are taken as the local mapping function; while the spatiotemporal chaotic system is a linear combination of two simple piecewise functions in Refs. [28] and [30].

(III) The key stream of the proposed algorithm in this paper is obtained by PMCM and a map between decimal number and binary sequence, with 256-bit; while that of the algorithms in Refs. [28] and [30] is set arbitrarily, with 128-bit, respectively.

(IV) The original message is padded with zeros to make the length a multiple of the length of each block message in Ref. [28]. In Ref. [30] and this paper, the padding rule is that the last padded 64 bits denote the length of the original message.

(V) Both the final hash value and each medial hash compression value of the proposed scheme are gained by mapping the state value of each lattice into a binary number with 4-bit in light of the rule in Table 1. But the hash values in Refs. [28] and [30] are produced by extracting the 32 bits after the decimal point from the state value of each site and then making the transformations of a certain rule.

5. Conclusion

In this paper, a novel algorithm for constructing a one-way hash function based on the spatiotemporal chaotic system is proposed and analysed. Not only the message block information but also its position in the whole message block chain serves as the initial conditions and key of the hash compression function, that is, the spatiotemporal chaotic system. The output of the hash compression function is composed of lattice state values by iterating the spatiotemporal chaos. In order to enhance the diffusion of the parallel processing of message blocks, we further compress the mixed result of all the hash compression values, then the final hash value is generated by mapping according to the rule in Table 1. Moreover, due to the parallel processing, the efficiency of the algorithm in practical application is improved to some extent. Simulation results show that the algorithm is extremely sensitive to to the message and the key, and has a strong diffusion capability and good collision resistance.

References

[1] Liu H J and Wang X Y 2010 *Comput. Math. Appl.* **59** 3320

[2] Cheddad A, Condell J, Curran K and McKevitt P 2010 *Opt. Commun.* **283** 879

[3] Liu H J and Wang X Y 2011 *Opt. Commun.* **284** 3895

[4] Wang X Y and Teng L 2011 *Nonlinear Dynamics* **67** 365

[5] Wang X Y and Jin C Q 2012 *Opt. Commun.* **285** 412

[6] Grzybowski JoseM V, Rafikov M and Macau ElbertE N 2010 *Acta Astronautica* **67** 881

[7] Wang X Y, Zhang N, Ren X L and Zhang Y L 2011 *Chin. Phys.* B **20** 020507

[8] Wu C J, Zhang Y B and Yang N N 2011 *Chin. Phys.* B **20** 060505

[9] Schneier B 1993 *Applied Cryptography: Protocols, Algorithms, and Source Code in C* (New York: John Wiley & Sons, Inc) p. 618

[10] Menezes A J, Van Oorschot P C and Vanstone S A 1996 *Hand Book of Applied Cryptography* (New York: CRC Press) p. 794

[11] Gao W, Li F and Wang X L 2009 *Computer Standards & Interfaces* **31** 282

[12] Rivest R L 1991 *Advances in Cryptology-CRYPTO'90* (Berlin: Springer-Verlag) p. 303

[13] Rivest R L 1992 *Request for Comments (RFC) 1321* (Internet Activities Board, Internet PrivacZ Task Force) April 1992

[14] Kou W D 1997 *Network Security and Standards* (Boston: Kluwer Academic)

[15] Wang X Y, Lai X J, Feng D G, Chen H and Yu X Y 2005 *Lecture Notes on Computer Science* **3494** 1

[16] Wang X Y and Yu H B 2005 *Lecture Notes on Computer Science* **3494** 19

[17] Wang X Y, Feng D G, Lai X J and Yu H B 2004 *Rump Session of Crypo'04* (Californi: Cryptology ePrint Archive)

[18] FIPS 180-1, *Secure Hash Standard*, NIST, US Department of Commerce, Washington D. C., April 1995

[19] Henzen L, Aumasson J P, Meier W and Phan R C W 2011 *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **19** 1746

[20] Wang X M, Zhang J S and Zhang W F 2003 *Acta Phys. Sin.* **52** 2737 (in Chinese)

[21] Peng F, Qiu S S and Long M 2005 *Acta Phys. Sin.* **54** 4562 (in Chinese)

[22] Zhang H, Wang X F, Li C H and Liu D H 2005 *Acta Phys. Sin.* **54** 4006 (in Chinese)

[23] Wang J Z, Wang Y L and Wang M Q 2006 *Acta Phys. Sin.* **55** 5048 (in Chinese)

[24] Zheng F, Tiao X J, Li X Y and Wu B 2008 *Chin. Phys.* B **17** 1685

[25] Yang Q T and Gao T G 2008 *Chin. Phys.* B **17** 2388

[26] Long M, Peng F and Chen G R 2008 *Chin. Phys.* B **17** 3588

[27] Xiao D, Liao X F and Wang Y 2009 *Neurocomputing* **72** 2288

[28] Ren H J, Wang Y, Xie Q and Yang H Q 2009 *Chaos, Solitons and Fractals* **42** 2014

[29] Wang X Y and Zhao J F 2010 *Neurocomputing* **73** 3224

[30] Wang S H and Shan P Y 2011 *Chin. Phys.* B **20** 090504

[31] Huang Z Q 2011 *Commun. Nonlinear Sci. Numer. Simul.* **16** 3245

[32] Li Y T, Deng S J and Xiao D 2011 *Neural Comput & Appl.* **20** 133

[33] Wang Y, Wong K W and Xiao D 2011 *Commun. Nonlinear Sci. Numer. Simul.* **16** 2810

[34] Kaneko K 1985 *Prog. Theor. Phys.* **74** 1033

[35] Bakhtiari S, Safavi-Naini R and Pieprzyk J 1996 *Lecture Notes in Computer Science* (Berlin: Springer-Verlag) **1029** 201

[36] Shannon C E 1949 *Bell System Technology Journal* **28** 656