Neural Comput & Applic (2013) 22:391–402
DOI 10.1007/s00521-011-0703-6

ORIGINAL ARTICLE

Improvement and performance analysis of a novel hash function based on chaotic neural network

Yantao Li · Di Xiao · Shaojiang Deng · Gang Zhou

Received: 12 March 2011 / Accepted: 5 July 2011 / Published online: 22 July 2011

Abstract In this paper, we reconsider and analyze our previous paper a novel hash algorithm construction based on chaotic neural network, then present equal-length and unequal-length forgery attacks against its security in detail, and then propose a significantly improved approach by utilizing a method of complicated nonlinear computation to enhance the security of the original hash algorithm. Theoretical analysis and computer simulation indicate that the improved algorithm can completely resist the two kinds of forgery attacks and also shows other better performance than the original one, such as better message and key sensitivity, statistical properties, which can satisfy the performance requirements of a more secure hash function.

Keywords Chaotic neural network · Forgery attack · Hash function · Nonlinear computation

1 Introduction

Hash function is a kind of one-way and compressive function, which can map any message with arbitrary length into a fixed length value and has been widely applied to integrity protection, message authentication and digital signature. Chaos has some inherent merits, such as one way, sensitivity to tiny changes in initial conditions and parameters, mixing property and ergodicity, which is suitable for constructing hash functions and has been used to design chaotic hash functions [1–11]. In 2003, Wong firstly proposed a chaotic hashing algorithm, which was built on the number of iterations of one-dimensional logistic map needed to reach the region corresponding to the character, along with a look-up table updated dynamically [1]. Then Zhang et al. [4] presented an n-dimensional chaotic dynamic system named feedforward-feedback nonlinear filter and then proposed a novel chaotic keyed hash algorithm in 2007. Fortunately, neural network also shows significant properties of confusion and diffusion, one way and compression, which is also adapted to build hash functions. Lian et al. [12] constructed a secure hash algorithm based on a three-layer network, where the three neuron-layers were used to realize data confusion, diffusion, and compression, and the multi-block hash mode was presented to support the plaintext with variable length in 2006. Consequently, it is significant and practical to combine chaos with neural network to create effective and more secure hash functions [13–15]. Based on chaos and neural network, namely, chaotic neural network, we proposed a novel hash algorithm, which greatly improved the spatiotemporal complexity of nonlinear dynamics [15]. However, according to recent works [16, 17], we find that our novel hash algorithm is vulnerable against forgery attack, in which the attacker tries to produce a message with a valid hash value based on three related messages and their corresponding hash values without the knowledge of the secret key.

Therefore, we reconsider and analyze our previous novel hash algorithm based on chaotic neural network [15], then present equal-length and unequal-length forgery attacks against its security in detail, and then propose a significantly improved approach by utilizing a method of

Y. Li (✉) · D. Xiao · S. Deng
College of Computer Science, Chongqing University, Chongqing 400044, China
e-mail: yantaoli@foxmail.com; yantaoli@cs.wm.edu; liyantao@live.com

Y. Li · G. Zhou
Department of Computer Science, College of William and Mary, Williamsburg, VA 23185, USA

complicated nonlinear computation to enhance the security of the original hash algorithm in this paper. Theoretical analysis and computer simulation indicate that the improved algorithm can completely resist the two kinds of forgery attacks and also shows other better performance than the original one, such as better message and key sensitivity, statistical properties, which can satisfy the performance requirements of a more secure hash function.

The rest part of this paper is organized as follows. Section 2 introduces the original hash algorithm and analyzes its security. In Sect. 3, the improved hash algorithm by utilizing a method of complicated nonlinear computation is described in detail. Performance analysis is illustrated in Sect. 4. Finally, conclusions are drawn in Sect. 5.

2 Original hash algorithm description and its security analysis

2.1 Description of the original hash algorithm

2.1.1 The chaotic maps

In the original hash algorithm, one-dimensional and piecewise liner chaotic map (PWLCM), one-dimensional and chaotic tent map (CTM), and 4-dimensional and one-way coupled map lattices (4D OWCML) are utilized, respectively.

The PWLCM proposed is defined as Eq. (1):

$$X(k+1)=f(X(k),P)=\begin{cases}X(k)/P, & 0\le X(k)<P,\\ (X(k)-P)/(0.5-P), & P\le X(k)<0.5,\\ (1-P-X(k))/(0.5-P), & 0.5\le X(k)<1-P,\\ (1-X(k))/P, & 1-P\le X(k)\le 1,\end{cases} \tag{1}$$

where $X\in[0,1]$ and $P\in(0,0.5)$ are the iteration trajectory value and control parameter of PWLCM, respectively.

The CTM utilized is expressed as Eq. (2):

$$X(k+1)=g(X(k),Q)=\begin{cases}QX(k), & 0\le X(k)<0.5,\\ Q(1-X(k)), & 0.5\le X(k)\le 1.\end{cases} \tag{2}$$

where $X\in[0,1]$ and $Q\in[\sqrt{2},2]$ are the iteration trajectory value and the parameter of the CTM, respectively.

The 4D OWCML employed is described as Eq. (3):

$$\begin{cases}x_1(k+1)=(1-\varepsilon)g(x_4(k))+\varepsilon g(x_3(k))\\ x_2(k+1)=(1-\varepsilon)g(x_1(k))+\varepsilon g(x_4(k))\\ x_3(k+1)=(1-\varepsilon)g(x_2(k))+\varepsilon g(x_1(k))\\ x_4(k+1)=(1-\varepsilon)g(x_3(k))+\varepsilon g(x_2(k))\end{cases} \tag{3}$$

where $x_1(0), x_2(0), x_3(0), x_4(0)\in[0,1]$ are the four initial values and the function $g()$ is the chaotic tent map (CTM), and $\varepsilon\in(0,1)$ is a coupling constant. The key generation function $gen()$ adopted in the algorithm based on 4D OWCML can be generalized as Eq. (4):

$$X(k+1)=gen(k+1)=(x_1(k+1)+x_2(k+1)+x_3(k+1)+x_4(k+1))/4. \tag{4}$$

2.1.2 The chaotic neural network

The chaotic neural network structure of the blocked hash function proposed in the algorithm is shown in Fig. 1, which consists of both the input layer and the output layer. The input layer has eight neurons, and the output layer has four neurons.

The input layer has eight neurons $W=(w_1,w_2,\ldots,w_8)$ and each neuron $w_i(i=1,2,\ldots,8)$ has eight input data $m_i(i=1,\ldots,8;9,\ldots,16;\ldots;57,\ldots,64)$. Each m_i has a length of 8 bits. The structure parameters of each input neuron are the same, including the weight $\omega=[\omega_1,\omega_2,\ldots,\omega_8]$, the bias β, and the transfer function $f()$ (PWLCM) with the parameter QI. For the input data $m_i(i=1,\ldots,8;9,\ldots,16;\ldots;57,\ldots,64)$, the corresponding eight neurons can be defined as $W=(w_1,w_2,\ldots,w_8)$.

$$w_i=f^{\tau}(\mathrm{mod}([\omega_1,\omega_2,\ldots,\omega_8]\times[m_{(i-1)\times8+1},m_{(i-1)\times8+2},\ldots,m_{(i-1)\times8+8}]^T+\beta,1),QI) \tag{5}$$

where τ is the iteration times of the PWLCM.

The output layer has four neurons $A=(a_1,a_2,a_3,a_4)$. The structure parameters of the output neurons are composed of the weight of each neuron $\omega_i=[\omega_{i,1},\omega_{i,2},\ldots,\omega_{i,8}](i=1,2,3,4)$, the four corresponding biases β_1, β_2, β_3, β_4, and the transfer function $f()$ with the

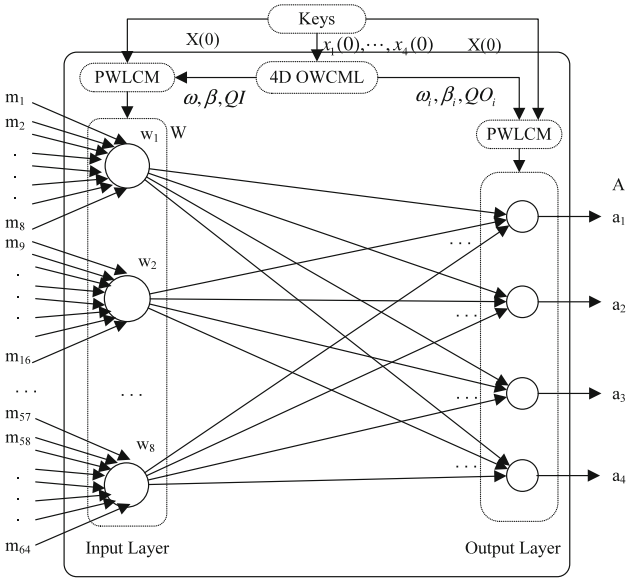

Fig. 1 The structure of two-layer neural network

parameter $QO_i(i = 1, 2, 3, 4)$. For the eight neurons $W = (w_1, w_2, \ldots, w_8)$ of the input layer, the corresponding four output neurons can be described as $A = (a_1, a_2, a_3, a_4)$.

$$a_i = f^{\tau}(\mathrm{mod}([\omega_{i,1}, \omega_{i,2}, \ldots, \omega_{i,8}] \times [w_1, w_2, \ldots, w_8]^T + \beta_i, 1), QO_i) \quad (6)$$

where τ is the iteration times of the PWLCM.

2.1.3 The original hash algorithm

The whole structure of the original hash algorithm can be illustrated in Fig. 2. Let $l = 128$ be the bit-length of the hash value. In advance, the original message *M'* with arbitrary length is padded with bits $(1010\ldots10)_2$ and the left 64-bit denoting the length of the original message M', such that its length is a multiple of *512*. The hash value with l bits is generated as follows.

1. Translate the appended message M into the corresponding ASCII code values and then partition M into p blocks: M_1, M_2,…,M_p, and each M_i $(i = 1,2,\ldots,p)$ has a length of 64-character (512-bit) denoted as $m'_{i,j}(i = 1, 2, \ldots, p; j = 1, 2, \ldots, 64)$. For each M_i, iterate the PWLCM $m'_{i,j}$ times with the initial value of last chaotic state $X(m'_{i,j-1})$ and the parameter $Q = \left(\frac{i}{p} + \frac{j}{64}\right)\Big/2$ to generate the corresponding decimal fraction $m_{i,j}(i = 1, 2, \ldots, p; j = 1, 2, \ldots, 64) \in [0, 1]$ which is short for $m_j(j = 1, 2, \ldots, 64)$ in Fig. 1. And as shown in Fig. 1, $m_j(j = 1, 2, \ldots, 64)$ is divided into eight groups for further process.
2. The secret key of the algorithm includes: the initial condition $X(0) \in [0, 1]$, initial parameter $P \in (0, 0.5)$, and iteration times τ of the PWLCM; the initial condition $X(0) \in [0, 1]$ and initial parameter $Q \in [\sqrt{2}, 2]$ of the CTM; the four initial values $x_1(0), x_2(0), x_3(0), x_4(0) \in [0, 1]$ and coupling constant $\varepsilon \in (0, 1)$ of 4D OWCML. They are set as: the iteration times $\tau = 45$ of the PWLCM; the parameter $Q = 1.52$ of CMT; 128-bit secret key is partitioned into four keys with length of 32 bits each and then transform the four into corresponding decimal fractions by means of linear transform to generate $x_1(0), x_2(0), x_3(0), x_4(0)$ and the coupling constant $\varepsilon = 1/4$ of 4D OWCML; the initial hash value $H(0) = \{0\}^l$. At the input layer, iterate 10 times of key generation function *gen*() (Eq. 4) to generate the weight $\omega = [\omega_1, \omega_2, \ldots, \omega_8]$, the bias β, and the parameter QI of the transfer function *f*() (Eq. 5), respectively. At the output layer, continue to iterate key generation function *gen*() 40 times to generate the weight $\omega_i = [\omega_{i,1}, \omega_{i,2}, \ldots, \omega_{i,8}](i = 1, 2, 3, 4)$, the biases $\beta_1, \beta_2, \beta_3, \beta_4$, and the parameter $QO_i(i = 1, 2, 3, 4)$ of the transfer function *f*()(Eq. 6),orderly and respectively.
3. At the input layer, compute Eq. (5) with the obtained parameters in step 2 to generate $W = (w_1, w_2, \ldots, w_8)$. At the output layer, compute Eq. (6) with the obtained parameters in step 2 and the output data of input layer to generate $A = (a_1, a_2, a_3, a_4)$.
4. Transform a_1, a_2, a_3, a_4 into the corresponding binary formats, then extract 32-bit of each and cascade these binary numbers orderly to generate the corresponding T_i.
5. After all the blocks are processed; the final hash value is generated by $H(M) = H(0) \oplus T_1 \oplus T_2 \oplus \cdots \oplus T_p$.

2.2 Theoretical analysis of forgery attack

Forgery attack means that the attacker tries to produce a message with a valid hash value without the knowledge of the secret key. In the original hash algorithm, the generation of T_i $(i = 1, 2,\ldots,p)$ is only related to the content and the order i of current message block M_i and the secret key. According to the finial hash value generation formula, $H(M) = H(0) \oplus T_1 \oplus T_2 \oplus \cdots \oplus T_p$, we can see that it is obtained through simple bit-wise exclusive OR (bit-wise XOR) operations among T_i $(i = 1, 2,\ldots,p)$, which leaves some flaw. In work [16], the authors proposed two different kinds of forgery attacks: equal-length forgery attack and unequal-length forgery attack, which are illustrated in the following, respectively.

On the one hand, in equal-length forgery attack, four equal-length messages (such as $M(1)$, $M(2)$, $M(3)$, $M(4)$) are padded and portioned into 512-bit blocks as follows.

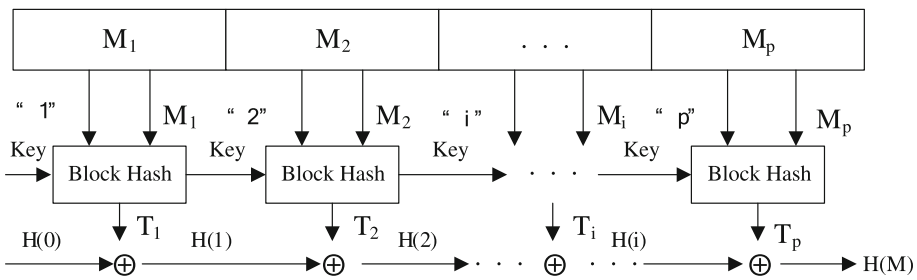

Fig. 2 The whole structure of the original algorithm

$$\begin{aligned} pad(M(1)) &= M_1M_2\cdots M_nM_{n+1}\cdots M_{p-1}M_p \\ pad(M(2)) &= M_1^{'}M_2^{'}\cdots M_n^{'}M_{n+1}^{'}\cdots M_{p-1}^{'}M_p \\ pad(M(3)) &= M_1^{'}M_2^{'}\cdots M_n^{'}M_{n+1}\cdots M_{p-1}M_p \\ pad(M(4)) &= M_1M_2\cdots M_nM_{n+1}^{'}\cdots M_{p-1}^{'}M_p \end{aligned} \tag{7}$$

where M_p denotes the padding block. Substituting Eq. (7) into $H(M) = H(0) \oplus T_1 \oplus T_2 \oplus \cdots \oplus T_p$, respectively, we can get the corresponding hash values of the four equal-length messages from original hash algorithm expressed as follows:

$$\begin{aligned} H(M(1)) &= H(0) \oplus T_1 \oplus T_2 \oplus \cdots \oplus T_n \oplus T_{n+1} \oplus \cdots \oplus T_{p-1} \oplus T_p \\ H(M(2)) &= H(0) \oplus T_1^{'} \oplus T_2^{'} \oplus \cdots \oplus T_n^{'} \oplus T_{n+1}^{'} \oplus \cdots \oplus T_{p-1}^{'} \oplus T_p \\ H(M(3)) &= H(0) \oplus T_1^{'} \oplus T_2^{'} \oplus \cdots \oplus T_n^{'} \oplus T_{n+1} \oplus \cdots \oplus T_{p-1} \oplus T_p \\ H(M(4)) &= H(0) \oplus T_1 \oplus T_2 \oplus \cdots \oplus T_n \oplus T_{n+1}^{'} \oplus \cdots \oplus T_{p-1}^{'} \oplus T_p. \end{aligned}$$

From above equations, we can derive the bit-wise XOR difference between $H(M(1))$ and $H(M(2))$ and that between $H(M(3))$ and $H(M(4))$ shown in Eq. (8):

$$\begin{aligned} &H(M(1)) \oplus H(M(2)) = H(M(3)) \oplus H(M(4)) \\ &= T_1 \oplus T_1^{'} \oplus T_2 \oplus T_2^{'} \oplus \cdots \oplus T_n \oplus T_n^{'} \oplus T_{n+1} \oplus T_{n+1}^{'} \oplus \cdots \oplus T_{p-1} \oplus T_{p-1}^{'}. \end{aligned} \tag{8}$$

Based on Eq. (8), we can obtain a simple but significantly computed result described in Eq. (9):

$$H(M(1)) \oplus H(M(2)) \oplus H(M(3)) \oplus H(M(4)) = 0. \tag{9}$$

That means the valid hash value of any one among the four messages can be derived from the corresponding hash values of the other three without the knowledge of the secret key.

On the other hand, in unequal-length forgery attack, four messages (such as $M(5)$, $M(6)$, $M(7)$, $M(8)$), where $M(5)$ has the same length and padding block with $M(6)$, and $M(7)$ has the same length and padding block with $M(8)$), are padded and portioned into 512-bit blocks as follows.

$$\begin{aligned} pad(M(5)) &= m_1m_2\cdots m_{p-1}m_p \\ pad(M(6)) &= m_1^{'}m_2^{'}\cdots m_{p-1}^{'}m_p \\ pad(M(7)) &= M_1\cdots M_{n-1}M_nM_{n+1}\cdots M_{np-n}M_{np-n+1}\cdots M_{np-1}M_{np} \\ pad(M(8)) &= M_1\cdots M_{n-1}M_n^{'}M_{n+1}\cdots M_{np-n}^{'}M_{np-n+1}\cdots M_{np-1}M_{np} \end{aligned} \tag{10}$$

where $M_{ni} = m_i$ and $M_{ni}^{'} = m_i^{'}(i \in 1, 2, \ldots, p-1)$, m_p and M_{np} represent the padding blocks. Substituting Eq. (10) into $H(M) = H(0) \oplus T_1 \oplus T_2 \oplus \cdots \oplus T_p$, respectively, we can also obtain the corresponding hash values of the four unequal-length messages from original hash algorithm expressed as follows:

$$\begin{aligned} H(M(5)) &= H(0) \oplus t_1 \oplus t_2 \oplus \cdots \oplus t_n \oplus t_{n+1} \oplus \cdots \oplus t_{p-1} \oplus t_p \\ H(M(6)) &= H(0) \oplus t_1^{'} \oplus t_2^{'} \oplus \cdots \oplus t_n^{'} \oplus t_{n+1}^{'} \oplus \cdots \oplus t_{p-1}^{'} \oplus t_p \\ H(M(7)) &= H(0) \oplus T_1 \oplus \cdots \oplus T_{n-1} \oplus T_n \oplus \cdots \oplus T_{np-n} \oplus \cdots \oplus T_{np-1} \oplus T_{np} \\ H(M(8)) &= H(0) \oplus T_1 \oplus \cdots \oplus T_{n-1} \oplus T_n^{'} \oplus \cdots \oplus T_{np-n}^{'} \oplus \cdots \oplus T_{np-1} \oplus T_{np}. \end{aligned}$$

From above equations, we can derive the bit-wise XOR difference between $H(M(5))$ and $H(M(6))$ and that between $H(M(7))$ and $H(M(8))$ shown in Eq. (11):

$$\begin{aligned} &H(M(7)) \oplus H(M(8)) \\ &= T_n \oplus T_n^{'} \oplus T_{2n} \oplus T_{2n}^{'} \oplus \cdots \oplus T_{np-n} \oplus T_{np-n}^{'} \\ &= t_1 \oplus t_1^{'} \oplus t_2 \oplus t_2^{'} \oplus \cdots \oplus t_{p-1} \oplus t_{p-1}^{'} \\ &= H(M(5)) \oplus H(M(6)). \end{aligned} \tag{11}$$

According to $M_{ni} = m_i$ and $M_{ni}^{'} = m_i^{'}(i \in 1, 2, \ldots, p-1)$, we can get $H(M_{ni}) = H(m_i)$ and $H(M_{ni}^{'}) = H(m_i^{'})$ and then deduce that $T_{ni} = t_i$ and $T_{ni}^{'} = t_i^{'}$. Based on Eq. (11), we can also obtain a simple but significantly computed result described in Eq. (12):

$$H(M(5)) \oplus H(M(6)) \oplus H(M(7)) \oplus H(M(8)) = 0. \tag{12}$$

We draw the same conclusions to equal-length forgery attack.

Therefore, according to the theoretical analysis, we can confirm that our original hash algorithm does not resist equal-length and unequal-length forgery attacks.

3 Improvement of the original hash algorithm

Since the forgery attack on the original hash algorithm depends on the fact that the final hash value $H(M) = H(0) \oplus T_1 \oplus T_2 \oplus \cdots \oplus T_p$ is obtained through simple bit-wise XOR operations with T_i ($i = 1, 2, \ldots, p$) corresponding to different message block M_i ($i = 1, 2, \ldots, p$), it is possible for the attacker to realize forgery attacks by constructing some special messages.

In order to overcome the above flaw, we propose an improved algorithm based on the original one and the whole structure of the improved algorithm is illustrated in Fig. 3. Compared with the original hash algorithm, there are two major improvements: One is to reprocess the

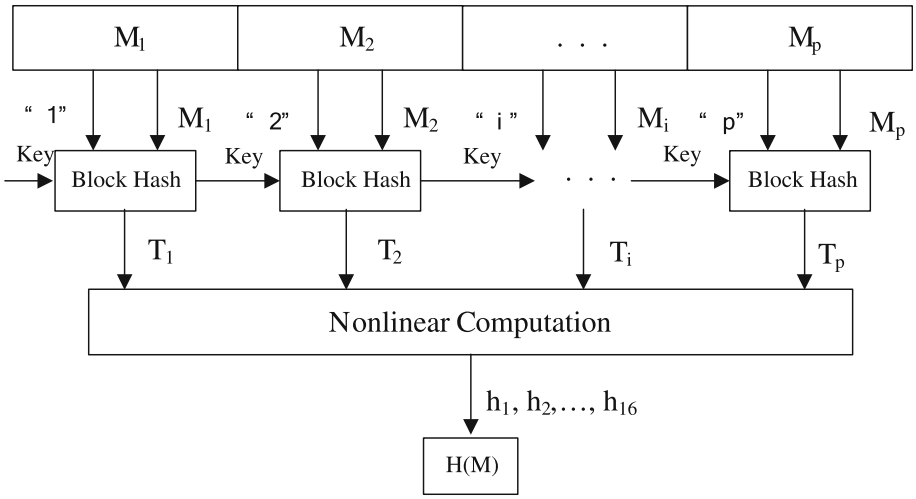

Fig. 3 The whole structure of the improved algorithm

obtained 128-bit T_i ($i = 1, 2,\ldots,p$) of each message block M_i in step 4, so that the data information of each T_i can be closely related by complicated Nonlinear Computation. The other is to generate the finial hash value directly by cascading 16 8-bit values obtained from the first improvement instead of $H(M) = H(0) \oplus T_1 \oplus T_2 \oplus \cdots \oplus T_p$ in step 5. The details of the improvements of the original hash algorithm are described as follows:

The first improvement is in step 4: based on the condition that the output values of Block Hash of each message block M_i ($i = 1,2,\ldots,p$) in Fig. 2 can be expressed as 128-bit T_i ($i = 1, 2,\ldots, p$), the 128-bit $T_i = (t_i^1 t_i^2 \cdots t_i^j \cdots t_i^{128})_2$ (transformed into binary numbers) is further divided into 16 8-bit integers, which are assigned as: $b_{i,1} = (t_i^1 t_i^2 \ldots t_i^8)_2$, $b_{i,2} = (t_i^9 t_i^{10} \ldots t_i^{16})_2, \ldots, b_{i,16} = (t_i^{121} t_i^{122} \ldots t_i^{128})_2$, where $i = 1,2,\ldots,p$. Therefore, after all the message blocks are processed, the corresponding output values, each of which is re-divided into 16 integers (transformed into decimal numbers) can be formatted as a $p \times 16$ matrix B, which can be formulated as Eq. (13):

$$B_{p\times 16} = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,16} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,16} \\ \vdots & \vdots & \ddots & \vdots \\ b_{p,1} & b_{p,2} & \cdots & b_{p,16} \end{bmatrix}. \tag{13}$$

Based on different element values in matrix $B_{p\times 16}$, 16 final 8-bit values h_1, $h_2,\ldots,h_{16}$ are generated by complicated Nonlinear Computation through the Eq. (14):

$$h_j = \bigoplus_{i=1}^{p} ((b_{i,j} \oplus row_i) + col_j) \quad (i = 1,2,\ldots,p; \quad j = 1,2,\ldots,16) \tag{14}$$

where "$\oplus$" denotes bit-wise exclusive OR operation, "+" represents addition modulo 2^8, and row_i ($i = 1,2,\ldots,p$) and col_j ($j = 1,2,\ldots,16$) are assigned by the Eqs. (15) and (16), respectively.

$$row_i = \frac{b_{i,1} + b_{i,2} + \cdots + b_{i,16}}{16} \quad (i = 1,2,\ldots,p) \tag{15}$$

$$col_j = \frac{b_{1,j} + b_{2,j} + \cdots + b_{p,j}}{p} \quad (j = 1,2,\ldots,16) \tag{16}$$

The other improvement is in step 5: transform the generated h_1, $h_2,\ldots,h_{16}$ into the corresponding binary formats and then cascade these binary numbers successively to generate the final hash value $H(M)$.

Therefore, in our improved hash algorithm, the final hash value has complicated nonlinear connections among different output values in the chaotic neural network of message blocks, which can effectively resist the forgery attacks.

4 Performance analysis

In this section, the forgery attack resistance will be analyzed in advance and other performance of the improved hash algorithm will also be proposed. The paragraph of the message applied in the following simulation experiments is chosen as [15]:

"*Chongqing University is a nationally famed comprehensive key university in China, directly under the State Ministry of Education, also a university listed among the first group of "211 Project" universities gaining preferential support in their construction and development from the Central Government of China. Currently, Chongqing University runs a graduate school and offers a wide range of undergraduate programs covering diverse branches of learning such as sciences, engineering, liberal arts, economics, management, law and education.*"

Table 1 Resistance against equal-length forgery attack

Secret keys (τ=45 of PWLCM, Q= 1.52 of CMT, 128-secret key, ε=1/4 of 4D OWCML)			
Message			Hash value (Hexadecimal formats)
$M(1)$	M_1	Chongqing University is a nation	08075450A6C7EA51E195D6291167A212
	M_2	ally famed comprehensive key uni	
$M(2)$	M_1'	versity in China, directly under	7420A9D8ED38C824E93109F9ADBC487C
	M_2'	the State Ministry of Education	
$M(3)$	M_1'	versity in China, directly under	491E0F5BAADDC9488BEFAE1E9F6A3181
	M_2	ally famed comprehensive key uni	
$M(4)$	M_1	Chongqing University is a nation	469466B6D70604368B0760B0B7A39A43
	M_2'	the State Ministry of Education	
$H(M(1)) \oplus H(M(2)) \oplus H(M(3)) \oplus H(M(4))$			7C27FD884BFF227508A4DFD0BCDBEA6E

Table 2 Resistance against unequal-length forgery attack

Secret keys (τ=45 of PWLCM, Q= 1.52 of CMT, 128-secret key, ε=1/4 of 4D OWCML)			
Message			Hash value (Hexadecimal formats)
$M(5)$	m_1	Chongqing University is a nation	7E707C708270807074807C7284867E80
$M(6)$	m_1'	ally famed comprehensive key uni	CA94D2D2AC32CAF252D2D40C326A728C
$M(7)$	M_1	versity in China, directly under	7420A9D8ED38C824E93109F9ADBC487C
	M_2	Chongqing University is a nation	
	M_3	the State Ministry of Education	
$M(8)$	M_1	versity in China, directly under	61A5461732198896F0D7D6D6C075B414
	M_2'	ally famed comprehensive key uni	
	M_3	the State Ministry of Education	
$H(M(5)) \oplus H(M(6)) \oplus H(M(7)) \oplus H(M(8))$			A161416DF1630A303FB47751DB25F064

4.1 Analysis of forgery attack resistance

The forgery attacks on the original hash algorithm are based on the utilization of the characteristic of bit-wise exclusive OR operation since the final hash value $H(M) = H(0) \oplus T_1 \oplus T_2 \oplus \cdots \oplus T_p$ is obtained through simple bit-wise exclusive OR operations with T_i ($i = 1, 2,\ldots,p$). In the improved hash algorithm, the improvements employ the complicated nonlinear connections among the different parts of output values of Hash Blocks, which directly generate the final hash value $H(M)$. Therefore, the Eqs. (8) and (11) do not work in the improved hash algorithm, which means that the improved hash algorithm can avoid the forgery attacks. The resistance experiments against equal-length and unequal-length forgery attacks have been done, respectively, to verify the resistance capability of the improved hash algorithm, and the corresponding results are listed in Tables 1 and 2, respectively. A careful observation of the experimental results in Tables 1 and 2 shows that the improved hash algorithm can effectively resist the two kinds of forgery attacks.

4.2 Statistical distribution of hash value

The uniform distribution of hash value is one of the most important properties of hash function, which is directly related to the security of hash function. Simulation experiment has been done on the paragraph of message.

Two 2-dimensional graphs are used to demonstrate the differences between the message and the final hash value. In Fig. 4a, since the ASCII codes of letters and symbols in ASCII Code Table are valued between 32 and 127, the message are localized within a small area, while in Fig. 4b, the hexadecimal hash value spreads around very irregularly. The similar experiment has been done to a special paragraph of ''blank space''-message with the same length as the above message. The contrast between message and hash value is demonstrated in Fig. 5. Even under this very extreme condition, the contrast is still distinct, and the distribution of hash value is irregular as well. The simulation results indicate that no information (including the statistic information) of the message can be left after the diffusion and confusion.

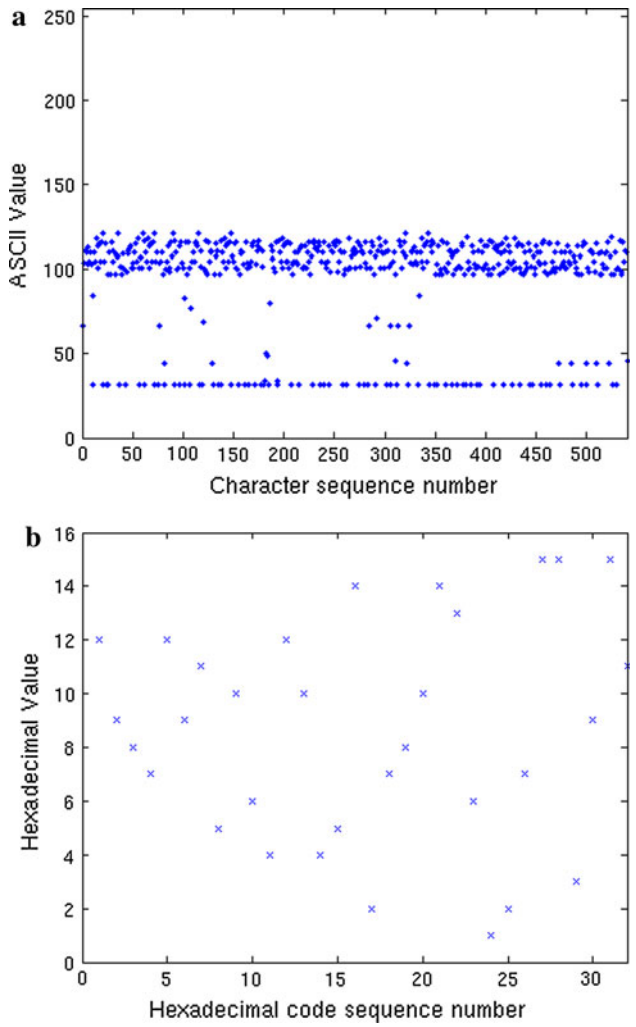

Fig. 4 Spread of messages and hash values: **a** distribution of the original message in ASCII code; **b** distribution of the hash values in hexadecimal format

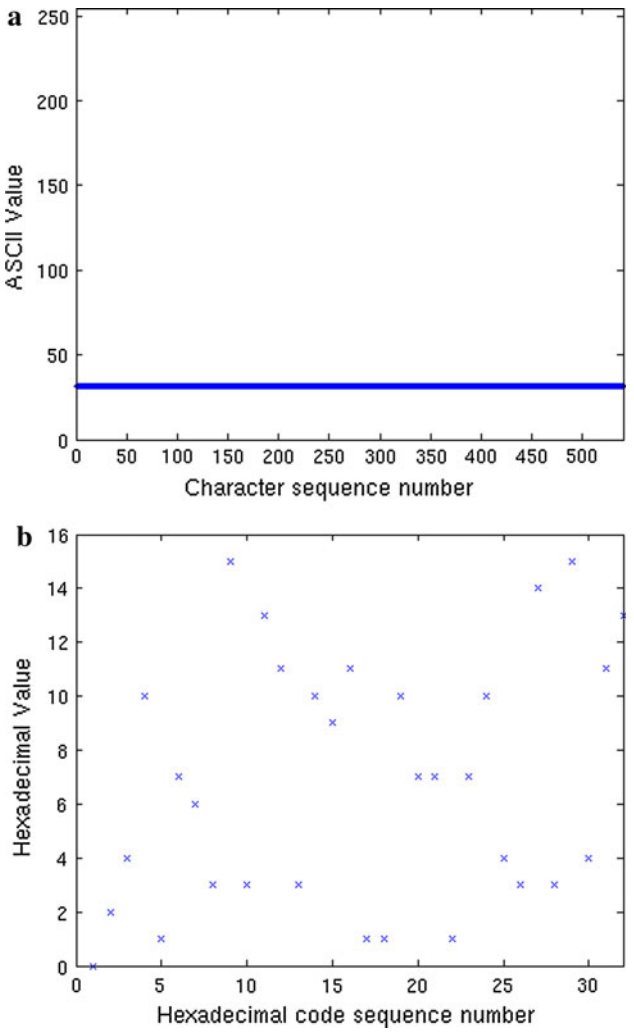

Fig. 5 Spread of all "blank space"-message and hash value: **a** distribution of all "blank space"-message; **b** distribution of the hash value in hexadecimal format

4.3 Sensitivity of hash value to the message and secret keys

In order to evaluate the sensitivity of hash value to the message and secret keys, hash simulation experiments have been conducted under the following different 7 conditions:

C1: The original message in this paper is the same as the one in original hash algorithm.
C2: Change the first character "*C*" in the original message to "*D*".
C3: Change the word "*directly*" in the original message to "*indirectly*".
C4: Add a blank space at the end of the original message.
C5: Change the parameter ε of the CMT from 1/4 to 1/3.
C6: Change τ of Eqs. (4) and (5) from 45 to 46.
C7: Exchange the first message block M_1-"*Chongqing University is a nationally famed comprehensive key uni*" with the second message block M_2-"*versity in China, directly under the State Ministry of Education*".

The corresponding hash values in hexadecimal formats are gotten from simulation experiments as follows, followed by the corresponding number of different bits compared with the hash value obtained under Condition 1:

C1: C987C9B5A64CA45E278AED6127FF39FB.
C2: A2BD0AC2D8F1E332CD64157FA8EAB23B (73)
C3: 6438ED4287C5DF117BA92EF7B52F570D (69)
C4: 24C8A81F7DFA810B481423D7ECAEAD3B (70)
C5: 770BAFB978EFCCB1EFF7F64A9F9CF8F3 (64)
C6: 66DEF3EB5EDD1A28DE49D7DE4CCF2DCF (71)
C7: FE2B7C9E4CB5A2F001F7E3C7C814BFB5 (72)

The corresponding graphical display of binary sequences is shown in Fig. 6.

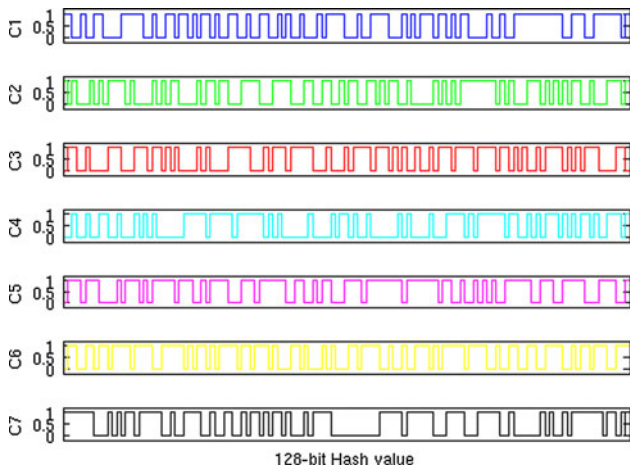

Fig. 6 Hash values under different conditions

The simulation result indicates that sensitivity property of the proposed algorithm is so perfect that any tiny difference of the message or key will cause huge changes in the final hash value. In addition, a careful analysis of sensitivity of hash value to secret keys in C5 and C6 reveals that tiny changes of secret keys cause corresponding 64-bit and 71-bit differences out of 128 bits, respectively. So we can confirm that the key factor determines the quality of the improved algorithm.

4.4 Statistical analysis of diffusion and confusion

Confusion and diffusion are two basic design criteria for encryption algorithm, including hash algorithms. Diffusion means spreading out of the influence of a single plaintext bit over many cipher text bits so as to hide the statistical structure of the plaintext. Confusion means the use of transformations that complicate dependence of the statistics of cipher text on the statistics of plaintext. Hash function requires the message to diffuse its influence into the whole hash space. This means that the correlation between the message and the corresponding hash value should be as small as possible. If the hash value is expressed in binary format, each bit can be only 0 or 1. Therefore, the ideal diffusion effect should be that any tiny change in the initial condition, control parameter or plaintext leads to a 50% changing probability for each bit of hash value. Four statistics used here are as follows: mean changed bit number B, mean changed probability P, standard deviation of the changed bit number ΔB and standard deviation ΔP. They are defined as: Mean changed bit number: $\overline{B} = \frac{1}{N}\sum_{i=1}^{N} B_i$; Mean changed probability: $P = (\overline{B}/l) \times 100\%$; Standard deviation of the changed bit number: $\Delta B = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(B_i - \overline{B})^2}$; Standard deviation: $\Delta P = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(B_i/l - P)^2} \times 100\%$, where N is the total number of tests, l is the length of the final hash value and B_i denotes changed bit number in the ith test.

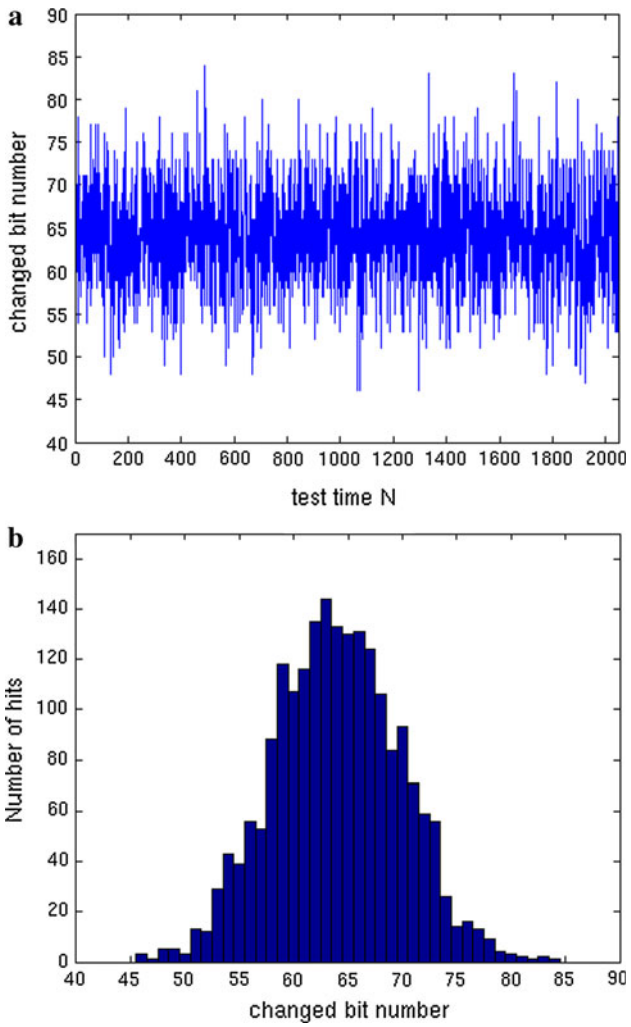

Fig. 7 Distribution of changed bit number: **a** Plot of B_i, **b** Histogram of B_i

The diffusion and confusion test is performed as follows: 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 obtained. Finally, two hash values are compared and the number of hanged bit is counted as B_i. This test is performed N times, and the corresponding distribution of changed bit number is shown as Fig. 7a, b, where $N = 2048$.

It can be seen from Fig. 7 a, b that the maximum changed bit number is 84 and the minimum is 46. It shows a good diffusion effect of the improved hash algorithm.

The same tests on the algorithm with $N = 256$, 512, 1,024 and 2,048 have also been performed. Under the condition that 1 bit is changed at each time, the corresponding values of $\overline{B}$, P, ΔB and ΔP are obtained as shown in Table 3.

Table 3 Statistics of number of changed bit

N	$N = 256$	$N = 512$	$N = 1024$	$N = 2048$	Mean
$\overline{B}$	63.6797	64.0293	64.0254	63.9756	63.9275
$P(\%)$	49.7498	50.0229	50.0198	49.9809	49.9433
ΔB	5.6429	5.7404	5.7115	5.8411	5.7340
$\Delta P(\%)$	4.4085	4.4847	4.4621	4.5634	4.4797

Based on the analysis of the data in Table 3, we can draw the conclusion: the mean changed bit number B and the mean changed probability P are both very close to the ideal value 64-bit and 50%. ΔB and ΔP are very little, which indicates the capability for diffusion and confusion is very stable. The statistical effect guarantees that attacker cannot carry out statistical attack.

4.5 Analysis of collision resistance

We have performed the following test to conduct quantitative analysis on collision resistance: first, the hash value for a paragraph of message randomly chosen is generated and stored in ASCII format. Then a bit in the paragraph is selected randomly and toggled and thus a new hash value is then generated and stored in the same format.

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. A plot of the distribution of the number of hits is given in Fig. 8. As seen from Fig. 8, there are 4 tests to hit twice, and 124 tests to hit once, while in 1920 tests, no hit occurs. The maximum number of equal characters at the same location in two hash values is only 4.

Moreover, the absolute difference of two hash values is calculated using the formula: $d = \sum_{i=1}^{N} \left| t(e_i) - t(e_i') \right|$,

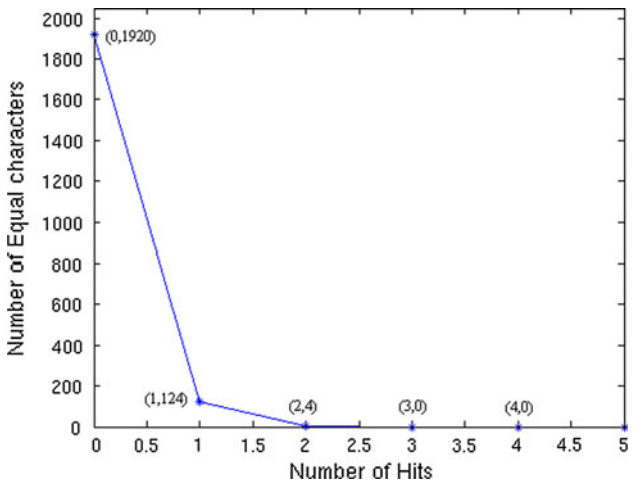

Fig. 8 Distribution of the number of ASCII characters with the same value at the same location in the hash value

Table 4 Absolute difference d of two hash values

Absolute difference	Maximum	Minimum	Mean	Mean/character
Values	2149	603	1319.5	82.4691

where e_i and e_i' are the ith ASCII character of the original and the new hash values, respectively, and the function $t(*)$ converts the entries to their equivalent decimal values. This kind of collision test is performed 2,048 times. The maximum, minimum, mean and mean/character values of d are listed in Table 4. The simulation result indicates that the sensitivity property of hash value is perfect that the absolute difference/character in the final hash value corresponding to any least difference of message or key will always wave around the theoretical value 85.3333, which is calculated as follows: if two hash values consist of independent and uniformly distributed random sequence, respectively, namely or e_i' has equal probability to be one integer of {0, 1, 2, 3,…, 254, 255}, and then the mean absolute difference/character is $1/3 \times 256 = 85.3333$.

4.6 Analysis of meet-in-the-middle resistance

Meet-in-the-middle attack means to find a contradiction through looking for a suitable substitution of any middle plaintext block. For instance, $M = (M_1,\ldots,M_i,\ldots,M_p)$ $(i = 2, 3,\ldots,p-1)$, the expected contradicted one is $M' = (M_1,\ldots,M_i',\ldots M_p)$. That is, the attack process is just to replace M_i with M_i' and keep the final hash value $H(M)$ unchanged. The corresponding simulation experiment is implemented as follows: replace the middle 512-bit message block M_i: "*y, Chongqing University runs a graduate school and offers a wide*" by the random message block named M_i' "*College of William and Mary is the second oldest college in USA*". The associated hash values of the original message *OM* and replaced message *RM* in

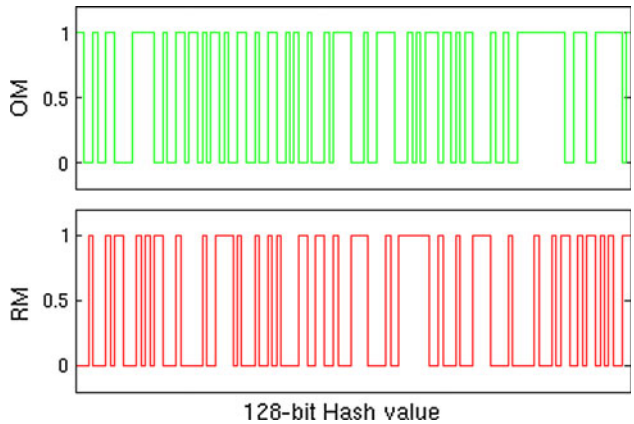

Fig. 9 Hash values under meet-in-the-middle resistance

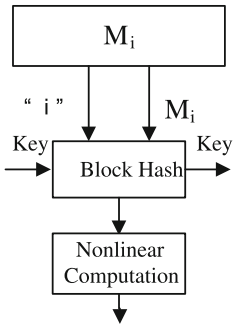

Fig. 10 Meet-in-the-middle attack

hexadecimal formats from the experiments are described in the following, followed by the number of different bits between *OM* and *RM* and the corresponding binary sequences depicted in Fig. 9.

OM: C987C9B5A64CA45E278AED6127FF39FB
RM: 1162B104F44A1991E13F911E104596A6 (76)

It follows from Fig. 9 that *RM* is obviously different from *OM*. In particular, there are 76-bit difference between *OM* and *RM*. Thus, the algorithm is against the attack.

A careful observation of Fig. 10 describing a random message block M_i ($i = 1, 2,\ldots, p$) chosen from the message reveals that the message block M_i denoted by the sequence number will be divided into 64 sub-blocks $m'_{i,j}(i = 1, 2, \ldots, p; j = 1, 2, \ldots, 64)$ and the ASCII code values and the order "*i, j*" of every sub-block are set as the iteration times and the parameter $Q = \left(\frac{i}{p} + \frac{j}{64}\right) \Big/ 2$ of the PWLCM to generate the corresponding 64 decimal fractions denoted as $m_{i,j}$ ($j = 1, 2,\ldots,64$), and after processed in Block Hash, T_i is generated, and through Nonlinear Computation, the elements of hash value about message block M_i are obtained. Any replacement definitely results in different ASCII code values and finally different hash value. Thus, this keeps the algorithm secure against this kind of attack.

Therefore, the improved algorithm is immune from meet-in-the-middle attack.

4.7 Comparison with other algorithms

Recently, some hash functions based on chaotic neural network [14, 15] and some hash functions based on chaos [4–11] have been presented. We choose some excellent algorithms from [7, 11, 14, 15] and the classical MD5 [5] as representatives and carry out the corresponding comparisons.

Table 5 The comparison of statistical performance

Statistics	Algorithms	$N = 256$	$N = 512$	$N = 1024$	$N = 2048$	Mean
$\overline{B}$	MD5 [5]	63.68	63.92	63.98	64.03	63.90
	Deng's [7]	63.97	63.52	63.73	63.84	63.77
	Wang's [11]	63.893	63.900	64.009	64.153	63.989
	Xiao's [14]	63.9453	64.1152	64.2666	64.0098	64.0842
	Li's [15]	63.7773	63.5605	63.5439	63.8076	63.6723
	This paper	63.6797	64.0293	64.0254	63.9756	63.9275
$P(\%)$	MD5 [5]	49.75	49.93	49.98	50.02	49.92
	Deng's [7]	49.98	49.63	49.79	49.88	49.82
	Wang's [11]	49.917	49.922	50.007	50.119	49.991
	Xiao's [14]	49.96	50.09	50.21	50.01	50.0675
	Li's [15]	49.8260	49.6567	49.6437	49.8497	49.7440
	This paper	49.7498	50.0229	50.0198	49.9809	49.9433
ΔB	MD5 [5]	5.38	5.78	5.73	5.66	5.64
	Deng's [7]	6.26	5.98	5.92	5.88	6.01
	Wang's [11]	5.367	5.491	5.671	5.767	5.574
	Xiao's [14]	5.4012	5.5437	5.7882	5.7236	5.6142
	Li's [15]	5.3915	5.6264	5.7143	5.7563	5.6221
	This paper	5.6429	5.7404	5.7115	5.8411	5.7340
$\Delta P(\%)$	MD5 [5]	4.20	4.36	4.48	4.42	4.37
	Deng's [7]	4.89	4.67	4.63	4.59	4.70
	Wang's [11]	4.193	4.289	4.430	4.506	4.355
	Xiao's [14]	4.22	4.33	4.52	4.47	4.385
	Li's [15]	4.2121	4.3956	4.4643	4.4971	4.3923
	This paper	4.4085	4.4847	4.4621	4.5634	4.4797

Table 6 The comparison of absolute differences d of two hash values

Absolute difference	Maximum	Minimum	Mean	Mean/character
MD5 [5]	2047	590	1304	81.5
Deng's [7]	2206	583	1399.8	87.49
Wang's [11]	2064	655	1367	85.44
Xiao's [14]	1952	605	1227.8	76.7375
Li's [15]	2220	687	1432.1	89.51
This paper	2149	603	1319.5	82.4691

Table 7 The comparison of maximum number of equal characters at the same location in the hash value

Algorithms	MD5 [5]	Deng's [7]	Wang's [11]	Xiao's [14]	Li's [15]	This paper
Values	2	2	2	2	2	2

4.7.1 *Comparison of statistics performance*

The corresponding statistical data from [7, 11, 14, 15] and MD5 [5] is tabulated in Table 5. The statistical data of this paper is also listed in Table 5 for convenient comparison. A careful observation of Table 5 shows that the statistical performance of all the hash algorithms is close to the ideal performance. It is worth to note that the mean changed bit number $\overline{B}$ and mean changed probability $P(\%)$ are the closest statistics to theoretical ideal values 64 bits and 50%, respectively, among all the proposed algorithms except Wang's algorithm [11], which means our improved algorithm can strongly resist statistical attacks.

4.7.2 *Comparison of collision performance*

The absolute difference d of the algorithms is obtained from [7, 11, 14, 15] and MD5 [5], which is tabulated in Table 6. As we know, the closer the absolute difference of each character is to the ideal value 85.3333, the stronger the collision resistance is. According to that, as shown in Table 6, we get the conclusion that the improved algorithm possesses a stronger collision resistance than all the proposed algorithms except Wang's algorithm [11]. The comparison of maximum number of equal characters at the same location in the hash value is shown in Table 7. All the maximum numbers in Table 7 of the proposed hash algorithm are only two, which means they all possess a strong collision resistance.

5 Conclusion

Reconsidering our past novel hash algorithm based on chaotic neural network, we present equal-length and unequal-length forgery attacks against its security in detail and then propose a significantly improved approach by utilizing a method of complicated nonlinear computation to enhance the security of the original hash algorithm in this paper. Theoretical analysis and computer simulation indicate that the improved algorithm can completely resist the two kinds of forgery attacks and also shows other better performance than the original one, such as better message and key sensitivity, statistical properties, which can satisfy the performance requirements of a more secure hash function.

Acknowledgments Our sincere thanks go to the anonymous reviewers for their valuable comments. The work described in this paper was fully funded by Project No. CDJZR10180003 supported by the Fundamental Research Funds for the Central Universities.

References

1. Wong KW (2003) A combined chaotic cryptographic and hashing scheme. Phys Lett A 307:292–298
2. Kwok HS, Tang WKS (2005) A chaos-based cryptographic Hash function for message authentication. Int J Bifur Chaos 15:4043–4050
3. Xiao D, Liao XF, Deng SJ (2005) One-way Hash function construction based on the chaotic map with changeable-parameter. Chaos Solitons Fractals 24:65–71
4. Zhang JS, Wang XM, Zhang WF (2007) Chaotic keyed Hash function based on feedforward–feedback nonlinear digital filter. Phys Lett A 362:439–448
5. Wang Y, Liao XF, Xiao D, Wong KW (2008) One-way hash function construction based on 2D coupled map lattices. Inform Sci 178:1391–1406
6. Xiao D, Liao XF, Deng SJ (2008) Parallel keyed hash function construction based on chaotic maps. Phys Lett A 372:4682–4688
7. Deng SJ, Li YT, Xiao D (2009) Analysis and improvement of a chaos-based hash function construction. Commun Nonlinear Sci Numer Simulat 15:1338–1347
8. Yang HQ, Wong KW, Liao XF et al (2009) One-way hash function construction based on chaotic map network. Chaos Solitons Fractals 41:2566–2574
9. Akhshani A, Behnia S, Akhavan A et al (2009) Hash function based on hierarchy of 2D piecewise nonlinear chaotic maps. Chaos Solitons Fractals 42:2405–2412

10. Xiao D, Shih FY, Liao XF (2010) A chaos-based hash function with both modification detection and localization capabilities. Commun Nonlinear Sci Numer Simulat 15:2254–2261
11. Wang Y, Wong KW, Xiao D (2011) Parallel hash function construction based on coupled map lattices. Commun Nonlinear Sci Numer Simulat 16:2810–2821
12. Lian SG, Sun JS, Wang ZQ (2006) Secure hash function based on neural network. Neurocomputing 69:2346–2350
13. Liu GL, Shan L, Dai YW et al (2006) One-way hash function based on chaotic neural network. Acta Phys Sin 55:5688–5706 (in Chinese)
14. Xiao D, Liao XF, Wang Y (2009) Parallel keyed hash function construction based on chaotic neural network. Neurocomputing 72:2288–2296
15. Li YT, Deng SJ, Xiao D (2011) A novel Hash algorithm construction based on chaotic neural network. Neural Comput Applic 20:133–141
16. Guo W, Wang XM, He DK et al (2009) Cryptanalysis on a parallel keyed hash function based on chaotic maps. Phys Lett A 373:3201–3206
17. Xiao D, Peng WB, Liao XF et al (2010) Collision analysis of one kind of chaos-based hash function. Phys Lett A 374:1228–1231