A method for designing Hash function based on chaotic neural network

Bo He[1], Peng Lei[2], Qin Pu[3], Zhaolong Liu[2]
[1]Key laboratory of electronic commerce and logistics, Chongqing University of Posts and Telecommunications, Chongqing, 400065, China
[2]College of Computer Science and Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China
[3]Chongqing Telecom Planning and Designing Institute Co., Ltd, Chongqing, 40041, China

***Abstract*—The neural network model has complex nonlinear behavior, which is very useful to design encryption algorithm and Hash function. In this paper, an algorithm for constructing one-way hash function based on chaotic neural network is proposed. The neural network model is initialized by two chaotic maps. Then, the message are divided into blocks with fixed length and inputted to neural network one by one. The final Hash value is extracted from status value of output layer cells. 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 provide data integrity.**

Keywords-Hash function; chaotic neural network; Cryptography; Information Security

I. INTRODUCTION

With the wide application of internet and computer technique, information security becomes more and more important. As we know, hash function is one of the cores of cryptography and plays an important role in information security. Hash function takes a message as input and produces an output referred to as a hash value. A hash value serves as a compact representative image (sometimes called digital fingerprint) of input string, and can be used for data integrity in conjunction with digital signature schemes [1]. Recent investigations on the collision frequencies reveal many undiscovered flaws in the well-known methods, such as MD5, SHA1, and RIPEMD [2, 3]. As a result, the research on designing hash functions with various methods attracts more and more attentions.

Chaos is a kind of deterministic random-like process found in nonlinear dynamical systems, which has some attractive features used to data protection, such as sensitive to initial value and ergodic. It is becoming a novel direction in constructing hash functions based on chaos. Based on Baptista's encryption method, Wong developed a scheme combining encryption and hashing [4]. Although it is able to encrypt messages and generate the corresponding hash value simultaneously, the efficiency and security of this scheme need further improvements [5]. Based on the piecewise linear chaotic map (PWLCM) or tent map, a hash algorithm with high efficiency is proposed [6]. To prevent attackers from breaking the hash function by predicting the chaotic series, complex chaotic systems are employed. A hash function based 2D coupled map lattices is present [7]. However, as the algorithms based on complex chaotic systems, its efficiency is not high. In recent years, some parallel hash functions based on complex dynamic systems are given [8-10]. But some security problems are found.

In this paper, we proposed a novel method for constructing hash function with the general iterative structure shown in Fig.1. The hash round function is designed by a chaotic neural network system. The chaotic neural network is sensitive to the input and status values, which guarantees the hash function owning high security performance. The remaining of this paper is organized as follows. The algorithm of hash function is described in section II. The performance of hash function is analyzed and compared with that of other algorithms in section III. Finally, conclusions are drawn in section IV.

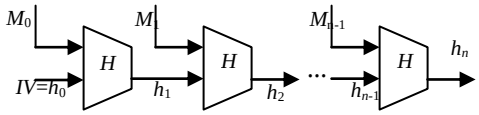

Figure 1. General iterative structure of hash function

II. THE SCHEME OF DESIGNING HASH FUNCTION

A. The chaotic neural network system

Compared to the simple chaotic map, chaotic neural network system has more complex dynamic behavior. When given the inner structure parameters and the input values, it is easy to compute output. Moreover, the output values of the system are sensitivity to the initial conditions and parameters. The inherent merits of chaotic neural network form the solid theoretical foundation for excellent Hash function construction.

The chaotic neural network system used in our scheme is shown in Fig. 2, which consists of two layers: the input layer and the out put layer. The input layer has 16 neurons, and each neuron has 256 input data. The structure parameters of each input neuron include the weights — $WI_{1,1}$, $WI_{1,2}$, ... $WI_{1,16}$, $WI_{2,1}$, $WI_{2,2}$, ..., $WI_{256,16}$, the biases — BI_1, BI_2, ..., BI_{16} and the transfer function—the logistic map with parameter μ. The output layer has eight neurons. The structure parameters of output neurons are the weights of each neuron — $WO_{1,1}$, $WO_{1,2}$, ... $WO_{1,8}$, $WO_{2,1}$, $WO_{2,2}$, ..., $WO_{16,8}$, the biases—BO_1, BO_2, ..., BO_8 and the transfer function—the logistic map with parameter μ.

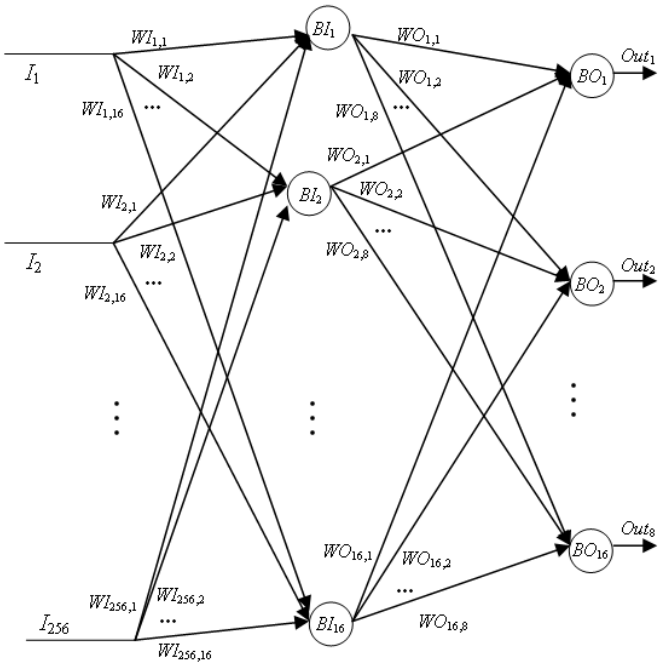

Figure 2. The chaotic neural network structure

B. Initialization of chaotic neural network

The 128-bit initial value *IV* are partitioned into 16 byte-values *IVs*. Denote *IVs*[*i*], *i* = 0, 1, 2, ..., 15 as the i^{th} byte value of *IV*. The *IVs* are used to set the initial values and status values of two chaotic maps shown in Eqs. (1) and (2). Then, the chaotic neural network is initialized by iterating these two chaotic maps.

$$x_{n+1} = \mu x_n (1 - x_n) \tag{1}$$

$$x_{n+1} = \begin{cases} x_i / b & 0 < x_i \le b \\ (1 - x_i)/(1 - b) & b < x_i < 1 \end{cases} \tag{2}$$

where $\mu \in [3.57, 4]$ and $b \in (0,1)$ are the control parameters.

1) Initialization of input weights and output weights. The process of initializing the weights of each input neuron and output neuron are described in the followng pseudocode fragments shown in Fig.3 and Fig.4, respectively.

```
1)  for(i=0; i<16; i++){
2)      x = (IVs[i]+0.1)/256
3)      Set x as the initial value of Eq.(1)
4)      Set μ = 4 and iterate Eq.(1) for 50 times;
5)      for(j = 0; j < 16; j++){
6)          iterate Eq.(1) only once;
7)          WI_{i,j} = x;
8)      }
8)  }
```

Figure 3. The pseudocode fragment of initializing input weights

```
1)  for(i=0; i<16; i++){
2)      b = (IVs[i]+0.1)/256;
3)      Set x = 0.1234 as the initial value of Eq.(2)
4)      Iterate Eq.(2) for 50 times;
5)      for(j = 0; j < 8; j++){
6)          iterate Eq.(2) only once;
7)          WO_{i,j} = x;
8)      }
9)  }
```

Figure 4. The pseudocode fragment of initializing output weights

2) Initialization of input biases and output biases. The process of initializing input biases and out weights are similar and the corresponding pseudocode fragments are given in Fig.5 and Fig.6, respectively.

```
1)  for(i=0; i<16; i++){
2)      x = (IVs[i]+0.1)/256;
3)      b = (IVs[i]+IVs[mod(i+1, 16)]+0.1)/512;
4)      iterate Eq.(2) for 50 times;
5)      BI_i = x;
6)  }
```

Figure 5. The pseudocode fragment of initializing input biases

```
1)  for(i=0; i<8; i++){
2)      x = (IVs[2i]+ IVs[2i+0.1])/512;
3)      b = (IVs[i]+IVs[i+8]+0.1)/512;
4)      iterate Eq.(2) for 50 times;
5)      BO_i = x;
6)  }
```

Figure 6. The pseudocode fragment of initializing output weights

C. The Hash scheme

Step 1. The original message *M* is padded such that its length is a multiple of *l* bits. Without loss of generality, we set *l* = 128.

Step 2. The padded message is partitioned into blocks M_0, M_1, ..., M_{n-1}, each has *l* bits.

Step 3. Set *IVs* = {0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef, 0xfe, 0xdc, 0xba, 0x98, 0x76, 0x54, 0x32, 0x10}. Each value in *IVs* is expressed in hexadecimal format. According to the rules described in *Section B*, initialize the parameters of chaotic neural network system with *IVs*.

Step 4. The message block M_0, M_1, ..., M_{n-1} are inputted into the chaotic neural network one by one. Each message block with 128-bit length is divided into 16 byte values. Then, the byte values are transformed to float numbers and used as the input variables of the neural network system.

Finally, adjusts the parameters of the neural network system. The details of processing one message block are described as below:

(1) Suppose the current message block is M_i and $m_{i,j}$ is the j^{th} byte value of M_i. $m_{i,j}$ is transformed to the corresponding float number $I_{i,j}$ according to Eq. (3).

$$I_{i,j} = (m_{i,j} + 0.1)/256 \qquad j = 0, 1, 2, \ldots, 15 \tag{3}$$

(2) Calculate the status value of input-layer neuron according to Eq.(4) and iterate the logistic map for 50 times to enhance the effect of input variables.

$$Nin_k = (BI_j + \sum_{j=0}^{15} WI_{j,k} \times I_{i,j}) \bmod 1 \qquad k = 0, 1, 2, \ldots, 15 \tag{4}$$

where Nin_k is the k^{th} status value of neuron. Then, obtain the status value of output-layer neuron according to Eq. (5) and also iterate the logistic map for 50 times.

$$Nout_k = (BO_j + \sum_{j=0}^{7} WO_{j,k} \times Nin_k) \bmod 1 \quad k = 0, 1, 2, \ldots, 7 \tag{5}$$

where $Nout_k$ is the k^{th} status value of output neuron.

(3) Adjust the biases and weights of output layer and the status value of input-layer neuron according to Eqs. (6), (7), and (8), respectively.

$$BO_k = (BO_k + Nout_k)/2 \qquad k = 0, 1, 2, \ldots, 7 \tag{6}$$

$$WO_{i,j} = \begin{cases} (WO_{i,j} + Nout_i)/2 & \text{if } j = 0 \\ ((WO_{i,j} + Nout_i)/2 + WO_{i,j-1})/2 & \text{if } j \neq 0 \end{cases} \tag{7}$$

$$\begin{cases} a_1 = \lfloor Nout_i \times 256 \rfloor \\ a_2 = \lfloor (Nout_i \times 256 - a_1) \times 256 \rfloor \\ Nin_i = (Nin_i + a_1/256)/2, \ i = 0, 2, 4, \cdots, 14 \\ Nin_i = (Nin_i + a_2/256)/2, \ i = 1, 3, 5, \cdots, 15 \end{cases} \tag{8}$$

Step 5. Transform the final status values of output neurons to the corresponding binary format and extract 16 bits (1^{st} to 16^{th} bits after the decimal point) from each neuron. Finally, juxtaposes these bits from left to right to get a 128-bit hash value.

III. PERFORMANCE ANALYSIS

A. Hash result of messages

We use the proposed algorithm to do hash simulation under the following five kinds of conditions:

Condition 1: The original message is "*With the wide application of internet and computer technique, information security becomes more and more important. As we know, hash function is one of the cores of cryptography and plays an important role in information security. Hash function takes a message as input and produces an output referred to as a hash value. A hash value serves as a compact representative image (sometimes called digital fingerprint) of input string, and can be used for data integrity in conjunction with digital signature schemes.*"

Condition 2: Changes the first Character *W* in the original message into *X*.

Condition 3: Changes the first word *with* in the original message into *without*.

Condition 4: Changes full stop at the end of the original message into comma.

Condition 5: Adds a blank space to the end of the original message.

The corresponding hash values in hexadecimal format are:

Condition 1:8C7AFBB127F4ECB412480101205CDF7C

Condition 2: E7623AB400F1091D0DA7CB9A3E77F29C

Condition 3: 119D14ADC14FA9DE0000FC215CB40B62

Condition 4: 438C986763A713D1C363F5CCF68B784A

Condition 5: 017CFFFEF879D121FFFFFF0C027D40EE

Based on the simulation result, it can be seen that any least difference of the message will cause huge changes in the final hash value.

B. Statistic analysis

From the view of binary format, Hash value consists of 1 and 0. So the ideal diffusion effect should be that any tiny changes in initial conditions lead to the 50% changing probability of each bit. In general, six statistics are used to evaluate the performance of Hash function, which are defined as follows:

Minimum changed bit number:

$$B_{\min} = \min(\{B_i\}_1^N) \tag{9}$$

Maximum changed bit number:

$$B_{\max} = \max\{B_i\}_1^N) \tag{10}$$

Mean changed bit number: $\bar{B} = \frac{1}{N}\sum_1^N B_i$ (11)

Mean changed probability: $P = (\bar{B}/128) \times 100\%$ (12)

Standard variance of the changed bit number:

$$\Delta B = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(B_i - \bar{B})^2} \quad (13)$$

Standard variance:

$$\Delta P = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(B_i/128 - P)^2} \times 100\% \quad (14)$$

where N is the total number of test and B_i is the number of changed bits in the i^{th} test.

We have performed the following test: A paragraph of message is randomly chosen and hash value is generated. Then a bit in the message is randomly selected and changed. The hash value of the changed message is generated. Finally two Hash values are compared. This kind of test is performed N times, and the corresponding distribution of changed bit number is shown as Fig. 7, where N = 2048.

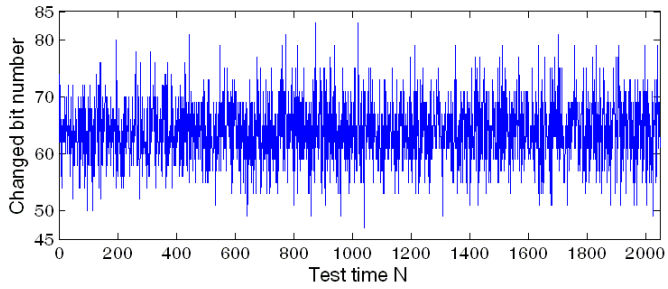

Figure 7. Distribution of changed bit number

TABLE I. THE STATISTICAL PERFORMANCE OF POPROSED HASH FUNCTION

***N* =**	**256**	**512**	**1024**	**2048**
$\bar{B}$	64.013	63.841	63.947	64.031
***P*(%)**	50.010	49.876	49.959	50.024
Δ *B*	5.545	5.629	5.673	5.608
Δ *P*(%)	4.301	4.411	4.417	4.406
B_{min}	50	49	46	45
B_{max}	79	81	82	83

Perform the tests with N = 256, 512, 1024, 2048, respectively, the corresponding data are listed in Table I. Based on the analysis of the data in Table I, we can draw the conclusion: the mean changed bit number $\bar{B}$ and the mean changed probability P are both very close to the ideal value 64 bit and 50%. Δ B and Δ P are very little, which indicates the statistical property of the Hash function is good and stable.

Moreover, similar test with MD5 is performed in [1], the result is shown in Table II. Based on Tables I and II, we can see that the proposed Hash function has almost the same performance with MD5.

TABLE II. THE STATISTICAL PERFORMANCE OF MD5

***N* =**	**256**	**512**	**1024**	**2048**
$\bar{B}$	64.683	64.437	64.205	64.060
***P*(%)**	50.534	50.341	50.160	50.046
Δ *B*	5.521	5.731	5.698	5.618
Δ *P*(%)	4.314	4.477	4.451	4.439
B_{min}	52	48	47	46
B_{max}	80	82	82	83

C. Collision analysis and test

1) Collision analysis. 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, the output neuron state 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 hash function is determined by the length of the hash value, which is 128-bit in our proposed function. According to the definition of birthday-attack, 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 neural network or extracting more bits from each neuron, a longer hash value can be obtained. Therefore, our hash function can resist this kind of attack.

2) Collision test. We perform the following test to do 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 message is selected randomly and changed. A new hash value is then generated and stored in ASCII format, too. The two hash values are compared and the number of ASCII character with the same value at the same location in the hash value is counted. This kind of collision test is performed 2048 times. The distribution of the number of ASCII characters with the same value at the same location in the hash value is given in Fig. 8. Notice that the maximum number of equal character is only 2 and the collision is very low.

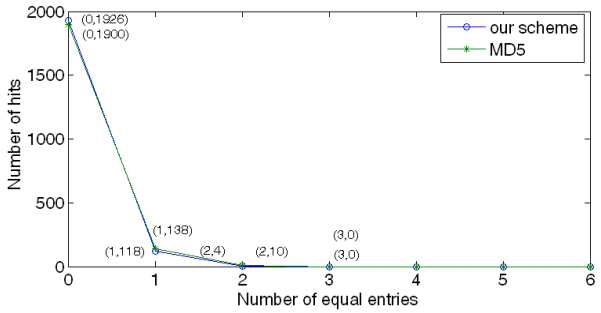

Figure 8. Distribution of the number of ASCII characters with the same value at the same location in the Hash value

Meanwhile the same tests are done with MD5. To the convenience of comparison, the distribution of the number of ASCII character with the smae value at the same location in the hash value is also shown Fig. 8. Obviously, the smaller the maximum number of equal characters is, the stronger the collision resistance is. Thus, our hash function has the same strong collision resistance as that of MD5.

3) Absolute difference. According Wong's method [4], the absolute difference of two hash values is calculated by using the following formula:

$$d = \sum_{i=1}^{N} \left| t(e_i) - t(e_i^{'}) \right| \tag{9}$$

where e_i and $e_i^{'}$ are the i^{th} ASCII character of the original and the new hash value, respectively. The function $t(\ )$ converts the entries to their equivalent decimal values. We calculate the maximum, mean, minimum values of d in the above 2048 times collision test. The results are listed in Table III. Meanwhile the same tests are done with MD5 and the results are also listed in Table III. It can be seen that the mean absolute difference of our Hash function is bigger than that of MD5. Hence our algorithm possesses a stronger collision resistance than MD5.

TABLE III ABSOLUTE DIFFERENCE OF OUR SCHEME AND MD5

Absolute difference	maximum	minimum	mean
Our scheme	2312	683	1504
MD5	2235	595	1390

IV. CONCLUSION

In this paper, a hash function based on chaotic neural network is proposed, which uses the general iterative structure of Hash function. The hash round function is constructed by a chaotic neural network. The chaotic neural network owns complex dynamic behavior. Since the output feedback model is employed, its output not only depends on the input and parameters of the neural network, but also its status. This dependence is enhanced by iterating the chaotic map, which is very useful to improve the performance of Hash function. Theory analysis and simulation tests show that the proposed algorithm fulfills the performance requirements of hash function. It is simple, efficient, practicable, and reliable. It is a good candidate for data integrity

ACKNOWLEDGEMENTS

The work described in this paper was supported by the National Natural Science Foundation of China (No. 61003256), the Postdoctoral Science Foundation of China (20110490082), the Foundation of Chongqing Education Committee (No KJ120506), the Natural Science Foundation of CQUPT (A2011-20) , and the Foundation of Chongqing Key Laboratory of Electronic Commerce and Logistics (No. ECML1007)

REFERENCES

[1] A. Menezes, P. van Oorschot, S. Vanstone, Handbook of applied cryptography, CRC Press, 1996.

[2] 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.

[3] X. Wang, H. Yu, How to break MD5 and other hash functions, in: Proceedings of Eurocrypt’ 05, Aarhus, Denmark, 2005, pp. 19-35.

[4] K. Wong, A combined chaotic cryptographic and hashing scheme, Physics Letters A 307 (2003) 292-298.

[5] G. Alvarez, F.Montoya, M. Romera, G. Pastor, Cryptanalysis of dynamic look-up table based chaotic cryptosystems, Physics Letters A 326 (2004) 211-218.

[6] X. Yi, Hash function based on chaotic tent maps, IEEE Transactions on Circuits and Systems II 52 (6) (2005) 354-357.

[7] Yong Wang, Xiaofeng Liao, Di Xiao, Kwok-Wo Wong. One-way hash function construction based on 2D coupled map lattices, Information Sciences. 2008 (178):1391-1406

[8] Di Xiao, Xiaofeng Liao, Yong Wang. Parallel keyed hash function construction based on chaotic neural network. Neurocomputing, Volume 72, Issues 10-12, June 2009, Pages 2288-2296.

[9] Di Xiao, Xiaofeng Liao, Yong Wang. Improving the security of a parallel keyed hash function based on chaotic maps. Physics Letters A. 373 (2009) 4346-4353.

[10] Yong Wang, Kwok-Wo Wong, Di Xiao. Parallel hash function construction based on coupled map lattices. Communications in Nonlinear Science and Numerical Simulation. 2011 (16) : 2810-2821.