Neural Comput & Applic (2011) 20:1305–1312
DOI 10.1007/s00521-011-0543-4

ORIGINAL ARTICLE

Parallel Hash function construction based on chaotic maps with changeable parameters

Yantao Li · Di Xiao · Shaojiang Deng · Qi Han · Gang Zhou

Received: 16 October 2009 / Accepted: 31 January 2011 / Published online: 17 February 2011

Abstract A parallel Hash algorithm construction based on chaotic maps with changeable parameters is proposed and analyzed in this paper. The two main characteristics of the proposed algorithm are parallel processing mode and message expansion. The algorithm translates the expanded message blocks into the corresponding ASCII code values as the iteration times, iterates the chaotic asymmetric tent map and then the chaotic piecewise linear map, continuously, with changeable parameters dynamically obtained from the position index of the corresponding message blocks, to generate decimal fractions, then rounds the decimal fractions to integers, and finally cascades these integers to construct intermediate Hash value. Final Hash value with the length of 128-bit is generated by logical XOR operation of intermediate Hash values. Theoretical analysis and computer simulation indicate that the proposed algorithm satisfies the performance requirements of a secure Hash function.

Keywords Chaos · Asymmetric tent map · Piecewise linear map · Parallel processing mode · Changeable parameter

1 Introduction

As one of the cores of cryptography, Hashing is a basic technique for information security. The conventional Hash functions such as MD4 [1], MD5 [2], and SHA-1 are realized through the complicated methods based on logical XOR operation or multi-round iterations of some available cipher. Since there are some defects in conventional Hash function constructions, the recently proposed chaos-based Hash functions exhibit an attractive design direction [3–15]. Until recently, based on Baptista's method [3] in 1998 that the message text was encrypted as the number of iterations applied in the chaotic map in order to reach the region correspondent to that text, Wong developed a Hashing scheme [4] in 2003, 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, and Xiao et al. created a one-way Hash function-based construction based on the chaotic map with changeable parameter [5] in 2005, which employed 3-unit iterations of one-dimensional chaotic piecewise linear map to generate final Hash value extracted some bits from each iteration, and in the same year, Yi [6] proposed a Hash function based on chaotic tent maps, which operated on a message with arbitrary length to produce $2l$-bit Hash value. In 2006, Lian et al. [7] employed a secure Hash function, which was constructed based on a three-layer network, where the three neuron-layers were used to realize data confusion, diffusion, and compression, respectively, and the multi-block Hash mode was presented to support the plaintext with variable length. Zhang et al. constructed an n-dimensional chaotic dynamic system named feedforward–feedback nonlinear filter (FFNF) and then proposed a novel chaotic keyed Hash algorithm using FFNF [8] in 2007. In 2008, Wang et al. [9] gave a one-way Hash function construction based on two-dimensional coupled map lattices, which employed a two-dimensional coupled map lattices with parameters leading to the largest Lyapunov exponent. In the same year, Yang

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

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

et al. [10] proposed a one-way Hash function construction based on chaotic map network. We employed a novel combined cryptographic and Hash algorithm based on chaotic control character [11] in 2009. In 2010, we proposed a novel Hash algorithm construction based on chaotic neural network [12] and also created Hash function based on the chaotic look-up table with changeable parameter [13]. However, all of the Hash algorithms mentioned above cannot be processed in a parallel mode, which can greatly improve the efficiency and speed of the Hash function.

It follows that a parallel Hash algorithm construction based on chaotic maps with changeable parameters is proposed and analyzed in this paper. The algorithm translates the expanded message blocks into the corresponding ASCII code values as the iteration times, iterates the chaotic asymmetric tent map and then the chaotic piecewise linear map, continuously, with the changeable parameters dynamically obtained from the position index of the corresponding message blocks, to generate decimal fractions, then rounds the decimal fractions to integers, and finally cascades these integers to construct intermediate Hash value. Final Hash value with the length of 128-bit is generated by logical XOR operation of intermediate Hash values.

The rest of this paper is arranged as follows: Section 2 introduces the preliminaries about the chaotic asymmetric tent map and piecewise linear map used in the proposed algorithm; In Sect. 3, the parallel Hash algorithm is described in detail; Performance is analyzed in Sect. 4; and Finally, conclusions are drawn in Sect. 5.

2 Preliminaries

2.1 Analysis of the chaotic asymmetric tent map

The one-dimensional chaotic asymmetric tent map in the algorithm is represented as (1):

$$x_{i+1} = \begin{cases} x_i/\alpha & 0 \le x_i \le \alpha \\ (1-x_i)/(1-\alpha) & \alpha < x_i \le 1 \end{cases} \quad (1)$$

where $x_i \in [0,1]$ and $\alpha \in (0,1)$ are the iteration trajectory value and the parameter of the chaotic asymmetric tent map, respectively. The map transforms an interval [0, 1] onto itself and contains only one parameter α. The map has some properties, which are suitable for constructing the Hash function, such as initial value sensitivity and parameter sensitivity. Figure 1 displays the chaotic iteration property with changeable parameter α valued in the interval (0, 1), if initial value $x_0 = 0.42223$. The simulation of the map iterating 50 times with the initial value $x_0 = 0.7654$ and parameter $\alpha = 0.6$ is shown in Fig. 2.

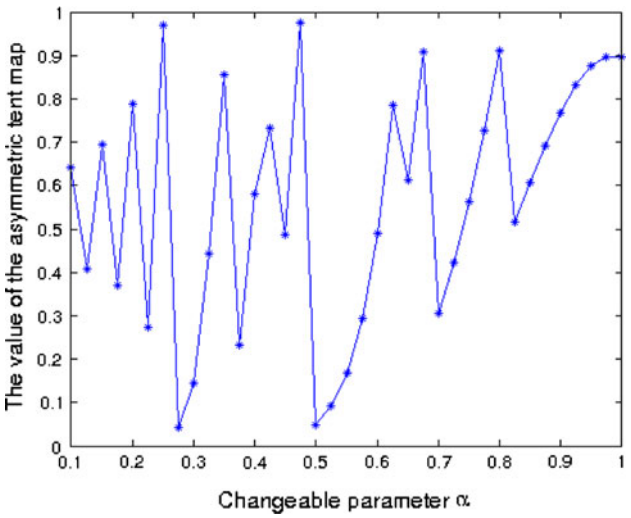

Fig. 1 Iteration property with changeable parameter α when $x_0 = 0.42223$

2.2 Analysis of the chaotic piecewise linear map

The one-dimensional and chaotic piecewise linear map in the algorithm is expressed as (2):

$$x_{i+1} = \begin{cases} x_i/\beta, & 0 \le x_i < \beta, \\ (x_i-\beta)/(0.5-\beta), & \beta \le x_i < 0.5, \\ (1-\beta-x_i)/(0.5-\beta), & 0.5 \le x_i < 1-\beta, \\ (1-x_i)/\beta, & 1-\beta \le x_i \le 1, \end{cases} \quad (2)$$

where $x_i \in [0,1]$ and β is the control parameter and belongs to (0, 0.5). The map is piecewise linear, and the parameter β ensures that the map runs in a chaotic state when $0 < \beta < 0.5$. It transforms an interval [0, 1] onto itself and contains only one parameter β. The chaotic piecewise linear map also has the same properties to chaotic asymmetric tent map that are fit for composing Hash function. Although the form of the map is simple and the equation involved is linear, with changeable parameter β valued in the interval (0, 0.5) and initial value $x_0 = 0.32323$, this map can display chaotic phenomena as shown in Fig. 3. Figure 4 shows the simulation of the chaotic piecewise linear map iterating 50 times with the initial value $x_0 = 0.32323$ and parameter $\beta = 0.4$.

3 Description of the parallel Hash algorithm

In this paper, the parallel Hash algorithm is developed by employing the high sensitivity to the initial conditions of a chaotic asymmetric tent map and a chaotic piecewise linear map. In order to improve the message sensitivity to the final Hash value, message expansion is introduced. Parallel processing mode and message expansion are the two main characteristics of the proposed Hash algorithm.

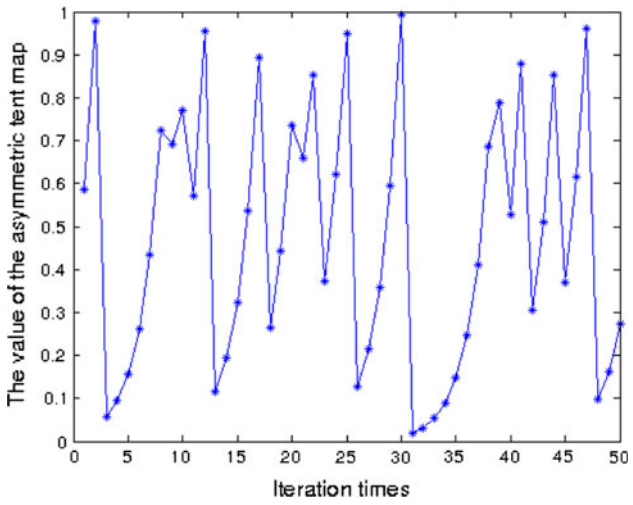

Fig. 2 Iteration property with $x_0 = 0.7654$ and parameter $\alpha = 0.6$

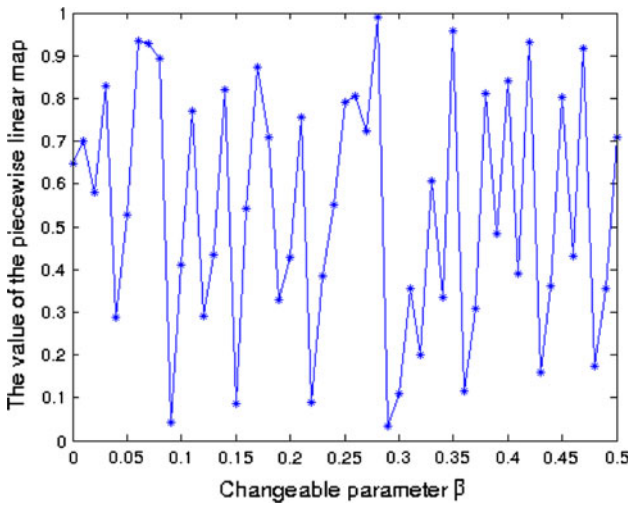

Fig. 3 Iteration property with changeable parameter β when $x_0 = 0.32323$

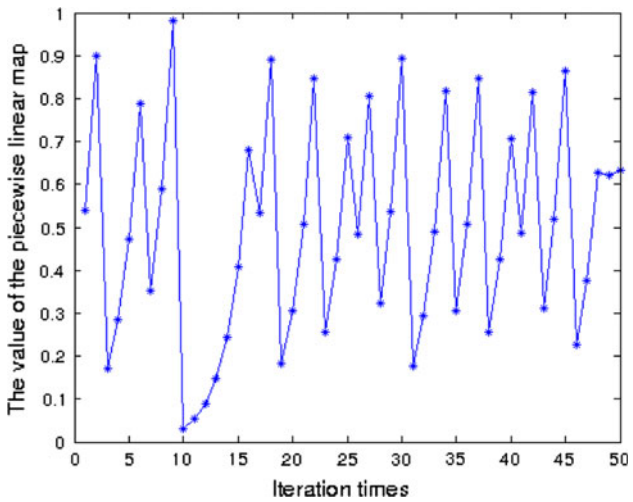

Fig. 4 Iteration property with $x_0 = 0.32323$ and parameter $\beta = 0.4$

3.1 Design of the parallel Hash algorithm

The main objective of a Hash function is to generate a fixed length Hash value from a message with arbitrary length, which plays an important role in data integrity protection, message authentication, and digital signature. Let $Hl = 128$ be the bit-length of Hash value in this paper, and the whole structure of Hash algorithm proposed is depicted in Fig. 5, which is composed of three steps: message expansion, parallel processing, and Hash value generation.

3.1.1 Step 1: message expansion

Message expansion is significant and necessary, because it greatly improves the sensitivity of each bit in original message to the final Hash value [16]. The plaintext is an arbitrary message that is expressed in a matrix M, for simple explanation of the extended message. Assume that M as shown in (3) is a $n \times 128$ plain message matrix, each element with a size of 8 bits.

$$M = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,127} & r_2 \\ m_{2,1} & m_{2,2} & \cdots & m_{2,127} & r_3 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ m_{n-1,1} & m_{n-1,2} & \cdots & m_{n-1,127} & r_n \\ c_{127} & c_{126} & \cdots & c_1 & r_1 \end{bmatrix} \tag{3}$$

A careful observation of message matrix $M_{n\times 128}$ shows that it is composed of matrix $M'_{n-1,127}$, r_i $(i = 1, 2,\ldots,n)$, and c_j $(j = 1, 2,\ldots,127)$. In the following, the generation of the three parts will be described, respectively.

As for the matrix $M'_{n-1,127}$, it is defined in (4) as follows:

$$M' = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,127} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,127} \\ \cdots & \cdots & \cdots & \cdots \\ m_{n-1,1} & m_{n-1,2} & \cdots & m_{n-1,127} \end{bmatrix} \tag{4}$$

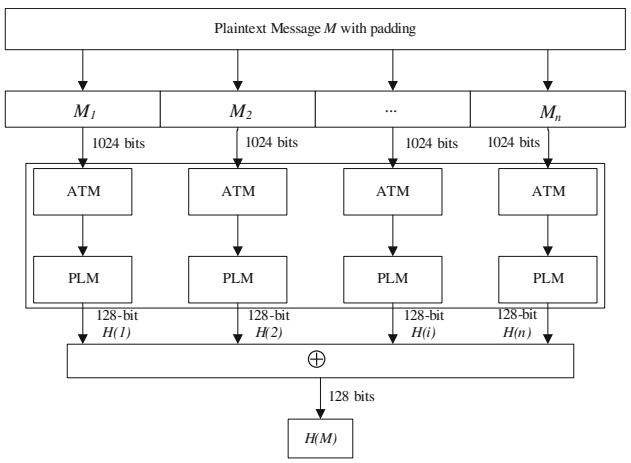

Fig. 5 The whole structure of Hash algorithm

where $M' = (m_{ij})$ is a (n−1) × 127 matrix composed of the original message with padding. The original message with arbitrary length firstly is padded with several bits, such that the length of padded message is a multiple of 127 × 8 = 1,016: let the length of original message be *om*, and then append *n* binary bits $(1010\ldots10)_2$, such that $(om + n)\ mod\ 1{,}016 = 1{,}016 - 64$, where $1 \le n \le 1{,}016$, and the left 64 bits are used to denote the length of the original message *om*. Secondly, the message is divided into 8-bit elements with number of a multiple of 127, each of which is then translated into its corresponding ASCII code value. Finally, all of the ASCII code values (decimal values) are assigned to m_{ij} ($i = 1, 2,\ldots, n-1; j = 1, 2,\ldots,127$), sequentially and respectively. That is the generation process of matrix $M'_{n-1,127}$ from padded original massage.

The other elements c_j ($j = 1, 2,\ldots,127$) and r_i ($i = 1, 2,\ldots,n$) in $M_{\,n \times 128}$ are defined in (5) and (6), respectively:

$$c_j = \bigoplus_{i=1}^{n} (m_{i,j} \oplus lm_i) + lm^j \quad (j = 1, 2, \ldots, 127) \tag{5}$$

where "$\oplus$" denotes bitwise exclusive OR operation, "+" represents addition modulo 2^8, and lm_i ($i = 1, 2,\ldots, n$) and lm^j ($j = 1, 2,\ldots, 127$) are the ($n + 127$) successive values multiplied by 2^8, which are obtained by iterating chaotic asymmetric tent map ($n + 127$) times with initial value $x_0 = 0.7654$ and parameter $\alpha = 0.6$, sequentially and respectively. c_j is a function operation on all the elements in the *j*th column of message matrix M' with generated numbers lm_i and lm^j.

$$r_i = \begin{cases} \sum_{j=1}^{127} ((m_{i,j} + lm^j) \oplus lm_i), & i = 1, 2, \ldots, n-1 \\ \sum_{j=1}^{127} ((col_j + lm^j) \oplus lm_i), & i = n \end{cases} \tag{6}$$

where "Σ" is summation modulo 2^8 and r_i is a function operation on all the elements in the *i*th row of the message matrix M' with generated numbers lm_i and lm^j.

For further convenient reference, set $M = (M_1, M_2,\ldots,M_n) = (m_{ij})$ ($i = 1, 2,\ldots, n; j = 1, 2,\ldots,128$), where M_i denotes the *i*th row elements of message matrix *M*.

3.1.2 Step 2: parallel processing

Since the parallel processing procedures are the same for each M_i ($i = 1, 2,\ldots, n$), as we can see from the whole structure of Hash algorithm (Fig. 5), for easy understanding, the processing procedure of message block M_i is randomly chosen as an example. For sub-blocks $m_{i,j}$ ($j = 1, 2,\ldots,128$) of the current M_i, iterate the chaotic asymmetric tent map (1) $\lfloor \frac{j}{Hl} \times m_{i,j} \rfloor$ times with initial value of the state value of last iteration of piecewise linear map $x_{plm}(m_{i,j-1})$ if $j \neq 1$ or $\frac{m_{i,128}}{256} \in [0, 1]$ if and only if $j = 1$ and changeable parameter $\alpha = (\frac{i}{n} + \frac{j}{Hl})/2$, to generate current state value of the asymmetric tent map $x_{acm}(m_{i,j})$, and then iterate the piecewise linear map (2) $\lfloor (1 - \frac{j}{Hl}) \times m_{i,j} \rfloor$ times with initial value $x_{acm}(m_{i,j})$ and changeable parameter $\beta = \frac{\alpha}{2}$, to generate the current value $x_{plm}(m_{i,j})$ of current sub-block $m_{i,j}$, and then round $x_{plm}(m_{i,j})$ to its nearest integer 0 or 1. Until all sub-blocks $m_{i,j}$ in M_i are processed, *Hl* numbers of 0 or 1 will be obtained. The intermediate *Hl*-bit Hash value $H(i)$ is generated by cascading the *Hl* numbers of 0 or 1.

3.1.3 Step 3: Hash value generation

After all the blocks are processed; the final Hash value is generated by $H(M) = H(1) \oplus H(2) \oplus \cdots H(i) \oplus \cdots \oplus H(n)$.

3.2 Characteristics of the parallel Hash algorithm

The proposed algorithm has two significant characteristics: the parallel processing mode and the improved sensitivity in message expansion, which are the two main contributions and advantages in this paper.

(1) Parallel processing mode

Since the presented Hash algorithm is mainly based on the iterations of the chaotic asymmetric tent map with initial value of last sub-block state value of chaotic piece linear map and changeable parameter α, and the iterations of chaotic piecewise linear map with last sub-block state value of asymmetric tent map as initial value and changeable parameter β, respectively, as we can see from Fig. 5, all message blocks M_i ($i = 1,2,\ldots,n$) are processed at the same time, which can greatly improve the efficiency and speed of the algorithm.

(2) Sensitivity improvement in message expansion.

Since the message matrix *M* consists of message $M'_{n-1,127}$, r_i ($i = 1, 2,\ldots, n$), and c_j ($j = 1, 2,\ldots, 127$), as shown in (3), each element of r_i is closely related to the *i*th row of $M'_{n-1,127}$ and each element of c_j is closely related to the *j*th column of $M'_{n-1,127}$ as well, according to (5) and (6). The message in r_i and c_j correlated to each message block greatly improves the sensitivity of the whole message *M*.

4 Performance analysis

4.1 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 following paragraph of message:

> 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.

Two 2-dimensional graphs are used to demonstrate the differences between the message and the final Hash value. In Fig. 6a, 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. 6b, 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. 7. 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.

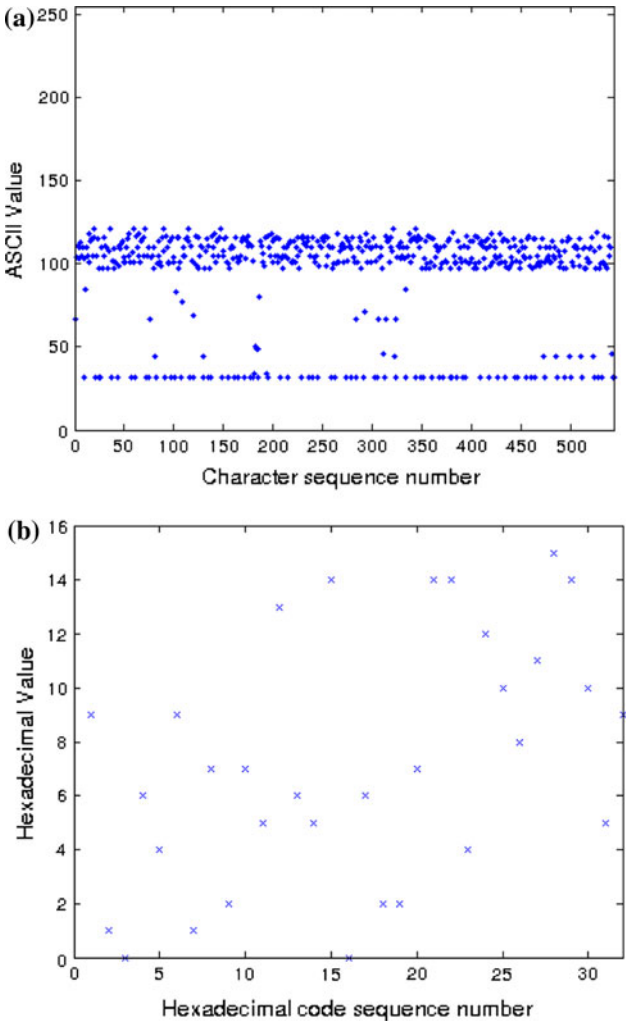

Fig. 6 Spread of message and Hash value: **a** distribution of the message in ASCII code; **b** distribution of the Hash value in hexadecimal format

4.2 Sensitivity of Hash value to the original message

In order to evaluate the sensitivity of Hash value to the message, Hash simulation experiments have been conducted under the following different six conditions:

C1: The original message is the same as the one in Sect. 4.1
C2: Change the first character '*C*' in the original message to '*c*'
C3: Change the word "*from*" in the original message to "*form*"
C4: Change the number "211" in the original message to "212"
C5: Add a blank space at the end of the original message
C6: Exchange the first message block *M1*-"*Chongqing University is a nationally famed comprehensive key uni*" with the second message block *M2*-"*versity in China, directly under the State Ministry of Education*"

The corresponding Hash values in hexadecimal formats are gotten from simulation experiments as follows:

C1: 91064917275D65E06227EE4CA8BFEA59.
C2: 8B36B1895D9095A59071C2A5A1F463ED.
C3: FE0A7D9256E2EB086227EE4CA8BFEA58.
C4: 9F0500A6BBFD1B791EC5898FAB05CEBF.
C5: 4735997205925656AA56654A8AC6ADB.
C6: 1A8391DB0595C62B05C60D837CD96623.

The corresponding graphical display of binary sequences is shown in Fig. 8. The simulation result indicates that sensitivity property of the proposed algorithm is so perfect that any tiny difference of the message will cause huge changes in the final Hash value.

4.3 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

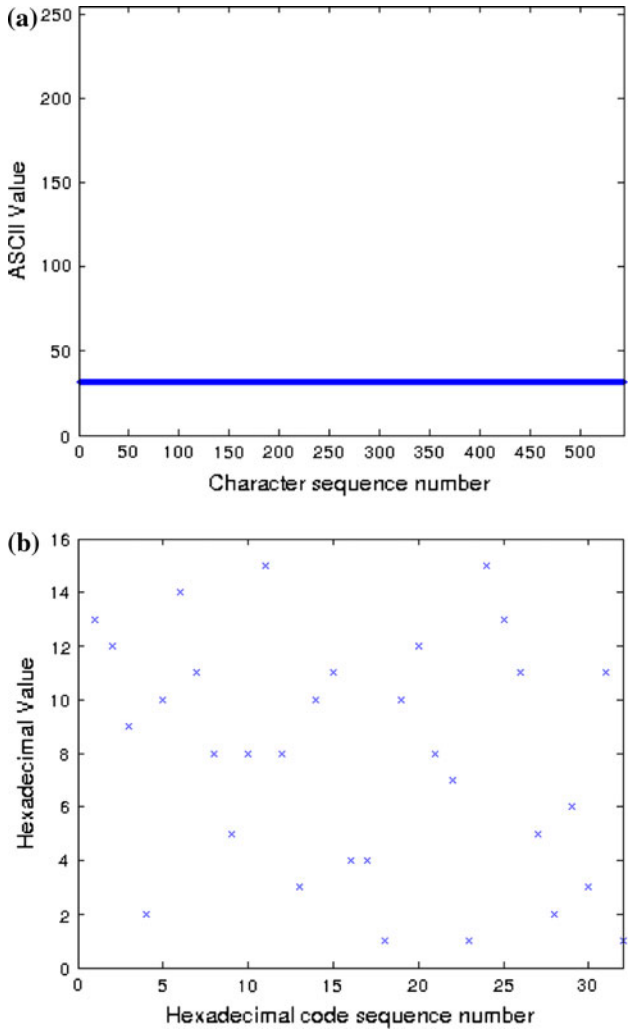

Fig. 7 Spread of all "blank space"—message and Hash value: **a** distribution of all "blank space"—message; **b** distribution of the Hash value in hexadecimal format

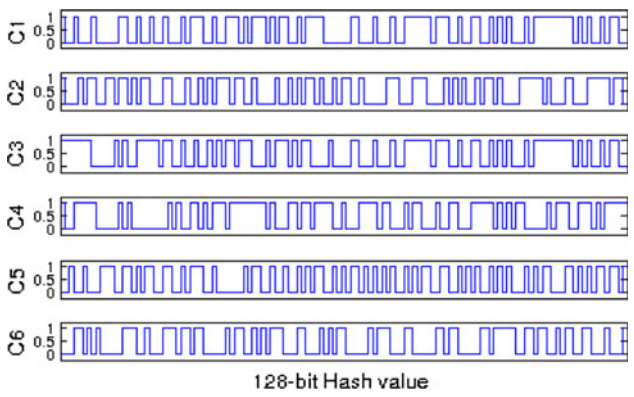

Fig. 8 Hash values under different conditions

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. For the Hash value in binary format, each bit can be only 0 or 1. Therefore, the ideal diffusion effect should be that any tiny change in the initial condition, control parameter, or plaintext leads to a 50% changing probability for each bit of Hash value. Usually, four statistics are defined as follows:

Mean changed bit number: $\overline{B} = \frac{1}{N}\sum_{i=1}^{N} B_i$

Mean changed probability: $P = (\overline{B}/Hl) \times 100\%$

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

where N is the total number of tests, B_i denotes changed bit number in the ith test, and Hl is the length of Hash value.

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. 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 in Fig. 9, where $N = 2{,}048$. Obviously, the changed bit number corresponding to 1 bit changed message concentrates around the ideal changed bit number 64-bit. It indicates that the algorithm has very strong capability for diffusion and confusion.

The same tests on the algorithm with $N = 256$, 512, 1,024, and 2,048 have also been performed, respectively. 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 1. Based on the analysis of the data in Table 1, we can see the mean changed bit number $\overline{B}$ and the mean changed probability P are both very close to the ideal value 64-bit and 50%. While ΔB and ΔP are very little, which indicate the capability for diffusion and confusion is very stable. Therefore, we can draw the conclusion: the statistical performance of the proposed algorithm approaches the ideal performance and satisfies the requirement of resisting statistical attacks.

4.4 Analysis of collision resistance

We perform the following test to conduct quantitative analysis on collision resistance [14, 15]: first, the Hash

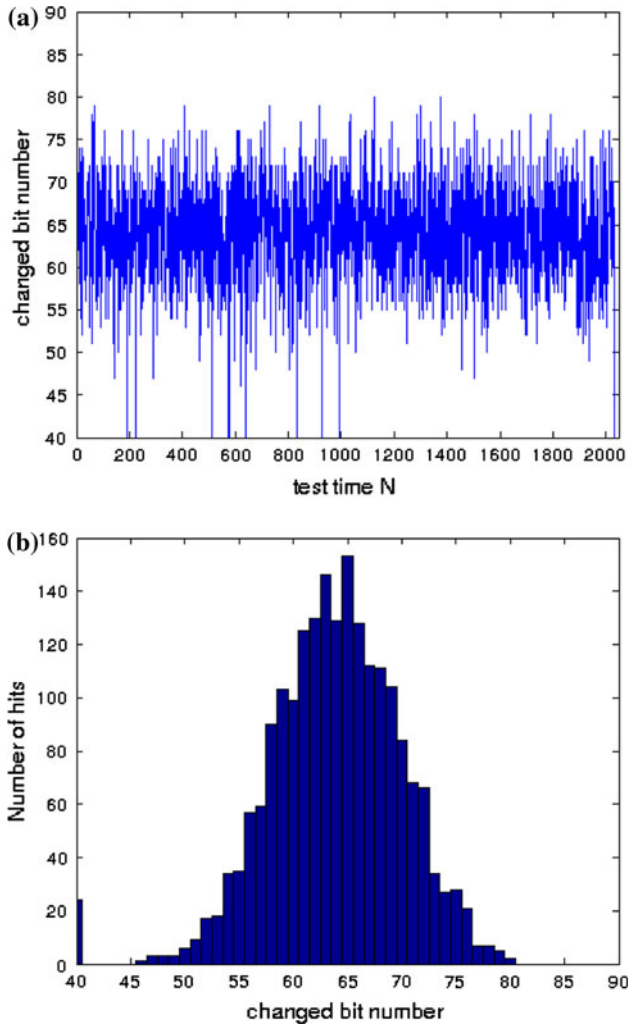

Fig. 9 Distribution of changed bit number: **a** plot of B_i, **b** histogram of B_i

Table 1 Statistics of number of changed bit

N	$N = 256$	$N = 512$	$N = 1{,}024$	$N = 2{,}048$	Mean
$\overline{B}$	63.8555	63.8398	63.6465	63.5684	63.7276
P (%)	49.89	49.87	49.72	49.66	49.79
ΔB	8.0024	7.3993	8.0226	7.4260	7.7126
ΔP (%)	6.25	5.78	6.27	5.80	6.03

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 values, namely, the number of hits, is counted. A plot of the distribution of the number of hits is given in Fig. 10, and we can see there are 25 tests hit twice and 124 tests hit once, while in 1,899 tests, no hit occurs. It is noticed that

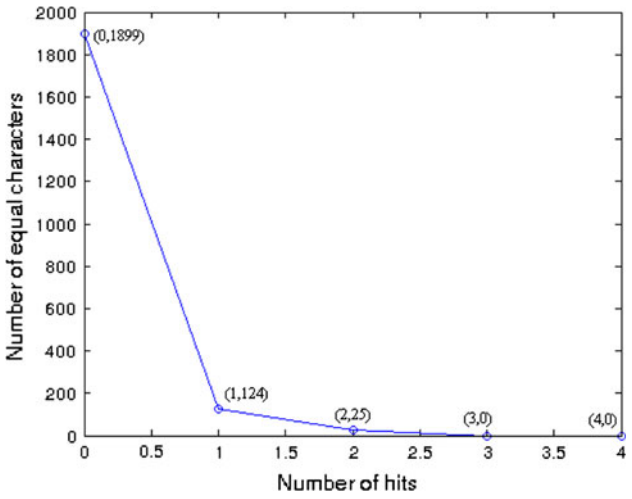

Fig. 10 Distribution of the number of ASCII characters with the same value at the same location in the Hash value

the maximum number of equal character is only 2 and the collision is very low. Moreover, the absolute difference of two Hash values is calculated using the formula: $d = \sum_{i=1}^{N} |t(e_i) - t(e_i')|$, where e_i and e_i' are the ith ASCII character of the original and the new Hash value, 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, mean, and minimum values of d are listed in Table 2.

4.5 Analysis of meet-in-the-middle resistance

Meet-in-the-middle attack [13–15] means to find a contradiction through looking for a suitable substitution of the last plaintext block. For instance, $M = (M_1, M_2, \ldots, M_{n-1}, M_n)$, the expected contradicted one is $M'' = (M_1, M_2, \ldots, M_{n-1}, M_n'')$. That is, the attack process is just to replace M_n with M_n'' and keep the final Hash value $H(M)$ unchanged. As a quantitative analysis, the corresponding simulation experiment is implemented as follows: replace the last message block M_n "*w and education.*" by the random message block named M_n'' "parallel acm plm". The associated Hash values of the original message and replaced message in hexadecimal formats from the experiments are described in the following, followed by the number of different bits between original message and replaced message, and corresponding binary sequences are shown in Fig. 11.

Original message: 91064917275D65E06227EE4CA8B FEA59.

Replaced message: 6BD8429DFCD30562C630DCDF 34A593C2 (63).

Table 2 Absolute difference of two Hash values

Absolute difference	Maximum	Minimum	Mean	Mean/character
Values	2,221	514	1367.6	85.4773

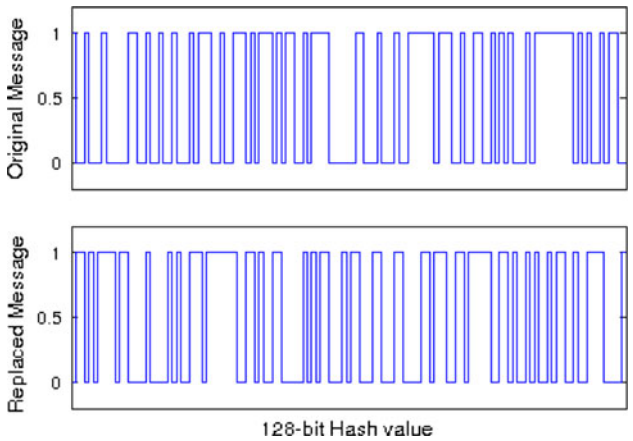

Fig. 11 Hash values under meet-in-the-middle resistance

It follows from Fig. 11 that the original message is obviously different from replaced message. In particular, there are 63-bit differences between original message and replaced message. Thus, the algorithm is against the attack.

A careful observation of Fig. 5 and Step 2 in Sect. 3 as a qualitative analysis reveals that both the initial value $x_0 = x_{plm}(m_{i,j-1})$and changeable parameter $\alpha = \left(\frac{i}{n} + \frac{j}{Hl}\right)/2$ of the chaotic asymmetric tent map and the initial value $x_0 = x_{acm}(m_{i,j})$ and changeable parameter $\beta = \left(\frac{i}{n} + \frac{j}{Hl}\right)/4$ of the chaotic piecewise linear map in the current sub-message block are closely related to the position index "i" and "j" of the message M. Therefore, it is very difficult to exchange M_n with M_n''. If it does as above instance, it will generate different Hash value. As a result, the proposed algorithm can resist meet-in-the-middle attack.

5 Conclusion

Based on the chaotic asymmetric tent map with changeable parameter and the chaotic piecewise linear map with changeable parameter, a parallel Hash algorithm construction is proposed and analyzed. There are two main characteristics of the algorithm: parallel processing mode and message expansion. The algorithm translates the expanded message blocks into the corresponding ASCII code values as the iteration times, iterates the chaotic asymmetric tent map and then the chaotic piecewise linear map, continuously, with the changeable parameters dynamically obtained from the position index of the corresponding message block, to generate decimal fractions, then rounds the decimal fractions to integers, and finally cascades these integers to construct intermediate Hash value. Final Hash value with the length of 128-bit is generated by logical XOR operation of intermediate Hash value. Theoretical analysis and computer simulation indicate that the proposed algorithm presents several interesting features, such as high message, good statistical properties, collision resistance, and secure against meet-in-the-middle attacks that can satisfy the performance requirements of Hash function.

Acknowledgments Our sincere thanks go to the anonymous reviewers for their valuable comments. The work described here was supported by the Fundamental Research Funds for the Central Universities (Grant No. CDJXS10182215), the National Natural Science Foundation of China (Grant Nos. 61070246, 61003247, 60873201), the Program for New Century Excellent Talents in University of China (NCET-09-0838, NCET-08-0603), the Natural Science Foundation Project of CQ CSTC (Grant Nos. 2010BB2047, 2009BB2211).

References

1. Rivest RL (1991) The MD4 message digest algorithm. Springer, Berlin, pp 303–311
2. Rivest RL (1992) The MD5 message digest algorithm. Request for Comments: 1321. MIT Laboratory for Computer Science and RSA Data Security, Inc
3. Baptista MS (1998) Cryptography with chaos. Phys Lett A 240:50–54
4. Wong KW (2003) A combined chaotic cryptographic and Hashing scheme. Phys Lett A 307:292–298
5. 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
6. Yi X (2005) Hash function based on chaotic tent maps. IEEE Trans Circuits Syst II Express Briefs 52:354–357
7. Lian SG, Sun JS, Wang ZQ (2006) Secure Hash function based on neural network. Neurocomputing 69:2346–2350
8. Zhang JS, Wang XM, Zhang WF (2007) Chaotic keyed Hash function based on feedforward–feedback nonlinear digital filter. Phys Lett A 362:439–448
9. 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
10. Yang HQ, Wong KW, Liao XF, Wang Y, Yang DG (2009) One-way Hash function construction based on chaotic map network. Chaos Solitons Fractals 41:2566–2574
11. 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
12. Li YT, Deng SJ, Xiao D (2011) A novel Hash algorithm construction based on chaotic neural network. Neural Comput Appl 20:133–141
13. Li YT, Xiao D, Deng SJ (2010) Hash function based on the chaotic look-up table with changeable parameter. Intern J Modern Phys B (In press)
14. Xiao D, Liao XF, Deng SJ (2008) Parallel keyed Hash function construction based on chaotic maps. Phys Lett A 372:4682–4688
15. Xiao D, Liao XY, Wong KW (2006) Improving the security of a dynamic look-up table based chaotic cryptosystem. IEEE Trans Circuits Sys II Express Briefs 53:502–506
16. Zhang CN, Lai CR (2004) A systematic approach for encryption and authentication with fault tolerance. Comput Netw 45:143–154