Analysis of One-way Alterable Length Hash Function Based on Cell Neural Network

Qun-ting Yang, Tie-gang Gao, Li Fan
College of Software, Nankai University, 300071, Tianjin, China
quntingyang@gmail.com
gaoetiegang@nankai.edu.cn
Fanlihml@163.com
Qiao-Iun Gu
Information and Technology College, Tianjin University of Technology and Education 300222, Tianjin, China
sjtuguqiaolun@yahoo.com.cn

Abstract—The design of an efficient one-way hash function with good performance is a hot spot in modern cryptography researches. In this paper, a hash function construction method based on cell neural network (CNN) with hyper-chaos characteristics is proposed. The chaos sequence generated by iterating CNN with Runge-Kutta algorithm, then the sequence iterates with every bit of the plaintext continually. Then hash code is obtained through the corresponding transform of the latter chaos sequence from iteration. Hash code with different length could be generated from the former hash result. Simulation and analysis demonstrate that the new method has the merit of convenience, high sensitivity to initial values, good hash performance, especially the strong stability, even if the hash code length is short relatively.

Keywords—cell neural network; one-way hash function; Hyper-chaos; hash length

I. INTRODUCTION

Chaos is a kind of deterministic random-like process generated by nonlinear dynamic systems. The properties of chaos includes: sensitivity to tiny changes in initial conditions, random-like behavior, ergodicity, unstable periodic orbits, desired diffusion and confusion properties, and one-way property. Neural network’s confusion and diffusion properties have been used to design encryption algorithms, such as the stream ciphers [1, 2] or block ciphers [3]. What is more, neural network has also the one-way property and is easy to design in circuit. The inherent merits of chaos and neural network form the theoretical foundation for hash function construction. The chaos [4, 5, 6] and neural network [7] have been used in the construction of one-way hash function, respectively. Recently, Liu [4] represented a hash scheme based on Henon iteration; Yi [8] proposed a way to construct chaotic-maps-based hash function; Xiao [6] introduce a way to construct Chebyshev chaos-based hash function; Wang [9] presented a construction algorithm based on mixed chaotic dynamic systems; Peng [5] proposed a hash function construction method based on two-dimensional hyper-Chaotic mapping; Lian [7] proposed a secure hash function based on neural network.

The one-way hash function construction algorithms above are all based on chaos or neural network. However, the chaos has defects such as the chaotic sequence generated by chaos may degenerates cycle sequence, while neural network model may be too simple to defend attacks. Thus, the combination of them may generate more complicated and secure algorithm. This paper presents a novel one-way hash function construction scheme based cell neural network with hyper-chaos characteristics. Simulations show that the proposed algorithm represents strong stability of diffusion and confusion comparing to algorithm [5, 6] above. It improves the performance of hash function chaos-based with comparative simplicity.

II. HASH FUNCTION CONSTRUCTION BASED ON CHAOTIC NEURAL NETWORK

A. The Chaotic Map Model

In the proposed algorithm, hyper-chaotic cellular neural network (HCCNN) used in this paper is modeled by the following system:

\[
\begin{align*}
  x_i &= -c_i + \sum_{j=1}^{4} w_{ij} v_j & i = 1, 2, 3, 4 \\
  v_i &= f_i(x_i)
\end{align*}
\]

(1)

where, \( f_i(x_i) = \frac{1}{2}(x_i + \|x_i - \|) \), \( W = (w_{ij}) \) is a \( 4 \times 4 \) matrix and \( w_{ij} = 0 \) for \( |i - j| > 1 \).

It has been shown that when matrix \( W \) to be

\[
W = \begin{pmatrix}
2.1 & 2.5 & 0 & 0 \\
-2.6 & 1 & 3 & 0 \\
0 & -2.8 & P & -1.1 \\
0 & 0 & 100 & 160
\end{pmatrix}
\]

(2)

The HCCNN (1) exhibits hyper-chaotic dynamics with two positive Lyapunov exponents with the parameter \( 0.27 < P < 0.67 \), for example, when \( P = 0.4 \), the four Lyapunov exponents are \( \lambda_1 = 0.125 \), \( \lambda_2 = 0.022 \), \( \lambda_3 = 0 \) and \( \lambda_4 = -95.95 \), respectively. The hyper-chaos attractors are shown in Fig.1.
B. The Description of One-way Hash Function Construction Algorithm

The principle of the construction algorithm is as follows: select four initial parameters $x_1(0)$, $x_2(0)$, $x_3(0)$ and $x_4(0)$, and then use Runge-Kutta formula to iterate HCCNN for $N_0$ times ($N_0$ is a constant integer more than 300 to avoid the harmful effect of transitional procedure). To make all of the contents of plaintext participate in the process of hash generation, some iteration of HCCNN are needed, and the times of iteration depend on the length of multimedia data, the detailed procedures are as follows:

Step 1: select four initial parameters $x_1(0)$, $x_2(0)$, $x_3(0)$ and $x_4(0)$. Where, the latter three could be got from the $x_1(0)$.

Step 2: Iterate CNN by using Runge-Kutta algorithm for $N_0$ times to avoid the harmful effect of transitional procedure, thus $x_1(N_0), x_2(N_0), x_3(N_0)$ and $x_4(N_0)$ are gotten.

Step 3: Then CNN is iterated continuously for $L/4$ times, where $L$ is the length of multimedia data and $L$ should be appended special characters such as “♦” if $L \% 4 \neq 0$.

Iteration formula can be described as

$$
\begin{align*}
    x_1(N_0+i) &= g_1(x_1(N_0+i-1)+B_4 \times (i-1)+1 \times 10^{-5}, \\
    x_2(N_0+i-1)+B_4 \times (i-1)+2 \times 10^{-5}, x_3(N_0+i-1)+B_4 \times (i-1)+3 \times 10^{-5}, x_4(N_0+i-1)+B_4 \times (i-1)+4 \times 10^{-5} \\
    x_2(N_0+i) &= g_2(x_2(N_0+i-1)+B_4 \times (i-1)+1 \times 10^{-5}, \\
    x_2(N_0+i-1)+B_4 \times (i-1)+2 \times 10^{-5}, x_3(N_0+i-1)+B_4 \times (i-1)+3 \times 10^{-5}, x_4(N_0+i-1)+B_4 \times (i-1)+4 \times 10^{-5} \\
    x_3(N_0+i) &= g_3(x_3(N_0+i-1)+B_4 \times (i-1)+1 \times 10^{-5}, \\
    x_2(N_0+i-1)+B_4 \times (i-1)+2 \times 10^{-5}, x_3(N_0+i-1)+B_4 \times (i-1)+3 \times 10^{-5}, x_4(N_0+i-1)+B_4 \times (i-1)+4 \times 10^{-5} \\
    x_4(N_0+i) &= g_4(x_4(N_0+i-1)+B_4 \times (i-1)+1 \times 10^{-5}, \\
    x_2(N_0+i-1)+B_4 \times (i-1)+2 \times 10^{-5}, x_3(N_0+i-1)+B_4 \times (i-1)+3 \times 10^{-5}, x_4(N_0+i-1)+B_4 \times (i-1)+4 \times 10^{-5}
\end{align*}
$$

where, $i = 1, 2, …, L/4$, $g_i(♦), i = 1, 2, 3, 4$ stands for Runge-Kutta iteration functions of $x_i, i = 1, 2, 3, 4$. After finishing the iteration, four decimal fractions $x_1, x_2, x_3, x_4$ will be generated.

Step 4: These decimal values are preprocessed firstly as follows:

$$
x_i = \text{de}2bi((\text{Abs}(x_i) – \text{Floor} (\text{Abs}(x_i))) \times 10^{14}) \quad (i = 1, 2, 3, 4)
$$

where, $\text{Abs}(x)$ returns the absolute value of $x$. $\text{Floor}(x)$ return the value of $x$ to the nearest integer less than or equal to $x$, and then converts them to binary value by using function de 2 bi( $x$) .

Step 5: Extract $N$ bits data from $x_i$. Then $4 \times N$ bits hash sequence will be generated. Hash results with variable length are got by extracting different $N$.

III. PERFORMANCE ANALYSIS AND TEST

A. Hash Results of Message

The plaintext followed is chosen randomly. We use the proposed algorithm to generate 128 bits hash information on the following message chosen at will under the condition of
$x_0 = 0.1234$. Then 64 and 32 bits length hash result was compressed from the 128 bits hash result.

The original message chosen is:

“Passage 2: American scientists have found that some birds are more intelligent than experts had believed. The scientists say birds have abilities that involve communication and different kinds of memory. In some unusual cases, their abilities seem better than those of humans.”

The hash result shows in hexadecimal format as followed:
128 bits: 967ED3227DCE345C31E97EA2E592FD09
64 bits: 6ED0FA6E59E0B4F1
32 bits: ACCAD86D

Very different hash result is got while the plaintext is changed very little. Take 128 bits hash result as an example. Different hash information is got under four kinds of condition followed.

Condition 1: Change the number “2” in the original message into “20”.
Condition 2: Change “i” of the second sentence in the original message into “t”.
Condition 3: Change the word “believed” in the original message into “thought”.
Condition 4: Append a blank at the end of the passage.

The corresponding Hash values are presented in hexadecimal format:
Original: 967ED3227DCE345C31E97EA2E592FD09
Condition 1: 743658BAD47A57CEF01E12FA451F877F
Condition 2: DEBDA568F1758F33CF8659978CD6031
Condition 3: E5EAD14698F2D5AE3942D95295ECB63C
Condition 4: 2C9CB565ADAF046E71069CFE0FFB68DB

The simulation results above indicate that the one-way property of the proposed algorithm is so perfect that any tiny change in initial conditions leads to the 50% changing probability of each bit.

We have performed the following diffusion and confusion test: A paragraph of message is randomly chosen and Hash value is generated; then a bit in the message is randomly selected and toggled and a new Hash value is generated. Variable hash results are got by modifying the way to process the initial parameters (take 32, 64 and 128 as examples).

Two Hash values are compared respectively under above two conditions. The number of changed bit is counted as $B_i$. This kind of test is performed $N$ (such as 256, 512, 1024 and 2408) times. The corresponding distribution of changed bit number is shown as Fig.3, where $N = 2048$ and the length of hash result is 64.

![Figure 3. Distribution of changed bit number and percent.](image)

Usually, four statistics are defined as follows:

$$\bar{B} = \frac{1}{N} \sum_{i=1}^{N} B_i$$

(5)

$$P = \left(\frac{\bar{B}}{128}\right) \times 100\%$$

(6)

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

(7)

$$\Delta P = \frac{1}{N-1} \sum_{i=1}^{N} \frac{B_i}{128} - P^2 \times 100\%$$

(8)

where, $N$ is total statistic number, $B_i$ is changed bit number when the $i$th simulation is performed. $\bar{B}$ and $P$ represent mean changed bit number and mean changed probability respectively. $\Delta B$ and $\Delta P$ indicate the stability of diffusion and confusion. Through the tests with $N = 256, 512, 1024, 2048$, respectively, the corresponding data are listed in Table 1.

<table>
<thead>
<tr>
<th>times</th>
<th>length</th>
<th>$\bar{B}$</th>
<th>$P(%)$</th>
<th>$\Delta B$</th>
<th>$\Delta P(%)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>32bits</td>
<td>15.98</td>
<td>49.93</td>
<td>0.18</td>
<td>3.13</td>
</tr>
<tr>
<td></td>
<td>64bits</td>
<td>31.36</td>
<td>49.00</td>
<td>0.25</td>
<td>3.07</td>
</tr>
<tr>
<td></td>
<td>128bits</td>
<td>63.44</td>
<td>49.56</td>
<td>0.43</td>
<td>3.11</td>
</tr>
<tr>
<td>512</td>
<td>32bits</td>
<td>16.01</td>
<td>50.02</td>
<td>0.13</td>
<td>2.22</td>
</tr>
<tr>
<td></td>
<td>64bits</td>
<td>31.45</td>
<td>49.14</td>
<td>0.18</td>
<td>2.18</td>
</tr>
<tr>
<td></td>
<td>128bits</td>
<td>63.48</td>
<td>49.59</td>
<td>0.30</td>
<td>2.20</td>
</tr>
</tbody>
</table>
\[ d = \sum_{i=1}^{N} |t(e_i) - t(e'_i)| \]  

(9)

Where \( e_i \) and \( e'_i \) is the \( i \)th ASCII character of the original and the new hash value, respectively; \( t() \) converts the entries to their equivalent decimal values. This kind of collision test is performed 10000 times with \( x_0(0) = 0.1234 \). The maximum, minimum, mean values of \( d \) is in Table II. A plot of the distribution of the number of ASCII characters with the same value at same location is given in Fig.5.

<table>
<thead>
<tr>
<th>d</th>
<th>Maximum</th>
<th>Minimum</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>32bits</td>
<td>590</td>
<td>19</td>
<td>280</td>
</tr>
<tr>
<td>64bits</td>
<td>1118</td>
<td>150</td>
<td>589</td>
</tr>
<tr>
<td>128bits</td>
<td>2025</td>
<td>537</td>
<td>1241</td>
</tr>
</tbody>
</table>

From table I, it indicates that one bit changing results almost 50% changing probability of all the hash results. While \( A_B \) and \( A_P \), indicating the stability of diffusion and confusion, are very little, representing the capability for diffusion and confusion is stable.

C. Sensitivity to Initial Parameter

The hash result is sensitive to the initial parameter, which is used to construct the chaos. Very different hash result is got while changing the initial parameter. Here the initial parameters \( x_0 \) are changed from 0.1234 to 0.2234, 0.3234 and 0.4234 in turn. It shows almost half of the bits will be changed along with the alteration of \( x_0 \). The 64 bits hash result used different \( x_0 \) is as followed, and the distribution of every bit is showed in Fig.4.

Original: 6ED0FA6E59E0B4F1  
Condition1: 48CE363DA319A2E5  
Condition2: 7DB26F1CCF26C791  
Condition3: 261E7F94FE5C9F86

<table>
<thead>
<tr>
<th>n (32)</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>32bits</td>
<td>16.01</td>
</tr>
<tr>
<td>64bits</td>
<td>31.42</td>
</tr>
<tr>
<td>128bits</td>
<td>63.45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n (64)</th>
<th>2048</th>
</tr>
</thead>
<tbody>
<tr>
<td>32bits</td>
<td>16.00</td>
</tr>
<tr>
<td>64bits</td>
<td>31.40</td>
</tr>
<tr>
<td>128bits</td>
<td>63.39</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>32bits</td>
</tr>
<tr>
<td>64bits</td>
</tr>
<tr>
<td>128bits</td>
</tr>
</tbody>
</table>

From table I, it indicates that one bit changing results almost 50% changing probability of all the hash results. While \( A_B \) and \( A_P \), indicating the stability of diffusion and confusion, are very little, representing the capability for diffusion and confusion is stable.

D. Analysis of Collision Resistance

Collision means that the hash results are identical to different random initial input. In order to investigate the collision resistance capability of the hashing approach producing hash results with different length, we have performed two collision tests, and 32bits, 64bits and 128bits hash results are investigated respectively in two tests [10].

In the first experiment, the hash value for a paragraph of message randomly chosen is generated and stored in ASCII format similarly. What is focused in this experiment is the possibility of colliding between every two hash results, thus every two hash results should be compared. The simulation is performed 10000 times equally in the same initial condition. The plot of the distribution of the number of ASCII characters with the same value at same location is given in Fig.6.

In the second experiment, the hash value for a paragraph of message randomly chosen is generated and stored in ASCII format similarly. What is focused in this experiment is the possibility of colliding between every two hash results, thus every two hash results should be compared. The simulation is performed 10000 times equally in the same initial condition. The plot of the distribution of the number of ASCII characters with the same value at same location is given in Fig.6.

Notice that: the maximum number of equal entries in the Fig.5 and Fig.6 are both 3 when hash length is 32 or 64; the maximum number is 4 when hash length is 128. So the hash results could resist collision well from the Fig.6.
E. Security of Key

In the algorithm proposed, \( x_0 \) are chosen as the secret keys, where \( x_0 \in [0,1] \) a decimal fraction. The length of the hash code could be chosen as \( H \) (such as 32, 64, 128). \( H \) is based on requirement and is always 128 bits. So the key space is huge enough to resist exhaustive key search.

F. Analysis of Speed

The executive speed of the algorithm proposed is proportional to the length of the plaintext nearly. The total iterative times is \( N_0 + \frac{L}{4} \), while \( N_0 \) is the initial iterative times, \( L \) indicates the length of the plaintext. So the iterative time required is very little when the plaintext is short.

IV. Conclusion

In this paper, a hash function construction method based on cell neural network with hyper-chaos characteristics is proposed and analyzed. The hash function adopts the inherent merits of chaos and neural network. And what’s more, hash results with different length shows well collision resistance using the hash function proposed. In the condition of less demanding, compressed hash code could be used for storage saving, if not, longer hash codes do well.

ACKNOWLEDGEMENT

This work was supported by the National Natural Science Foundation of China (No.60873117).

REFERENCES


