Chaos, Solitons and Fractals 91 (2016) 639–648

Contents lists available at ScienceDirect

Chaos, Solitons and Fractals

Nonlinear Science, and Nonequilibrium and Complex Phenomena

journal homepage: www.elsevier.com/locate/chaos

Chaotic hash function based on circular shifts with variable parameters

Yantao Li [a,b,*], Xiang Li [a]

[a] College of Computer and Information Sciences, Southwest University, Chongqing 400715, PR China
[b] State Key Laboratory for Novel Software Technology, Nanjing University, Jiangsu 210023, PR China

ARTICLE INFO

Article history:
Received 18 May 2016
Revised 27 August 2016
Accepted 28 August 2016

Keywords:
Chaos
Hash function
Variable parameters
Piecewise linear chaotic map
One-way coupled map lattice

ABSTRACT

We propose a chaotic hash algorithm based on circular shifts with variable parameters in this paper. We exploit piecewise linear chaotic map and one-way coupled map lattice to produce initial values and variable parameters. Circular shifts are introduced to improve the randomness of hash values. We evaluate the proposed hash algorithm in terms of distribution of the hash value, sensitivity of the hash value to slight modifications of the original message and secret keys, confusion and diffusion properties, robustness against birthday and meet-in-the-middle attacks, collision tests, analysis of speed, randomness tests, flexibility, computational complexity, and the results demonstrate that the proposed algorithm has strong security strength. Compared with the existing chaotic hash algorithms, our algorithm shows moderate statistical performance, better speed, randomness tests, and flexibility.

© 2016 Elsevier Ltd. All rights reserved.

1. Introduction

A hash function is a one-way function that can be used to map arbitrary length of digital data to digital data with fixed length. The digital data returned by a hash function are referred to as hash value, or message digest. Hash functions have been used in a wide range of applications, such as integrity protection [1], message authentication [2] and digital signature [3], which have the properties of sensitivity to initial conditions, diffusion and confusion, collision resistance. Traditional hash functions such as MD5 and SHA-1 are mainly based on logical operations, modular arithmetic operations or digital algebraic operations, which greatly impact the security, since attacks on these algorithms have been discovered [4–9]. For instance, X.Y. Wang found an effective method to reduce the complexity of collisions of SHA-1, issued as a Federal Information Processing Standard by NIST [5]. Chaos has some inherent merits of one way, sensitivity to tiny modifications in initial conditions and parameters, mixing property and ergodicity, which can be used for designing chaotic hash functions. K.W. Wong is the first to propose the chaotic hash function, which is built on the number of iterations of one-dimensional logistic map needed to reach the region corresponding to the character, along with a lookup table updated dynamically [10]. After then, chaotic hash functions are gradually attracting more and more researchers to study ranging from the use of simple maps, such as tent map [11–13] and logistic map [14,15], to the use of more complicated maps of the sine map [16], standard map [17], piecewise linear or nonlinear chaotic maps [18–21], and high-dimensional chaotic maps [22–24].

Since these chaotic maps in cryptanalytic studies reveal security weakness [25–33], we propose a circular shift based chaotic hash algorithm with variable parameters in this paper. We exploit piecewise linear chaotic map and one-way coupled map lattice to produce initial values and variable parameters. Circular shifts are introduced to improve the randomness of hash values. We evaluate the proposed hash algorithm in terms of distribution of the hash value, sensitivity of the hash value to slight modifications of the original message and secret keys, confusion and diffusion properties, robustness against birthday and meet-in-the-middle attacks, collision tests, analysis of speed, randomness tests, flexibility, computational complexity, and the results demonstrate that the proposed algorithm has strong security strength. Compared with the existing chaotic hash algorithms, our algorithm shows better statistical performance and strong collision resistance.

The remainder of this paper is organized as follows: Section 2 briefly introduces the preliminaries of piecewise linear chaotic map and one-way coupled map lattice used in our algorithm. In Section 3, we design the circular shift based chaotic hash function with variable parameters in detail, which is composed of parameter initialization, message processing and hash value generation. We excessively evaluate the performance of the proposed hash algorithm in Section 4 and present conclusions in Section 5.

* Corresponding author.
E-mail address: yantaoli@foxmail.com, liyantao@live.com, yantaoli@swu.edu.cn (Y. Li).

http://dx.doi.org/10.1016/j.chaos.2016.08.014
0960-0779/© 2016 Elsevier Ltd. All rights reserved.

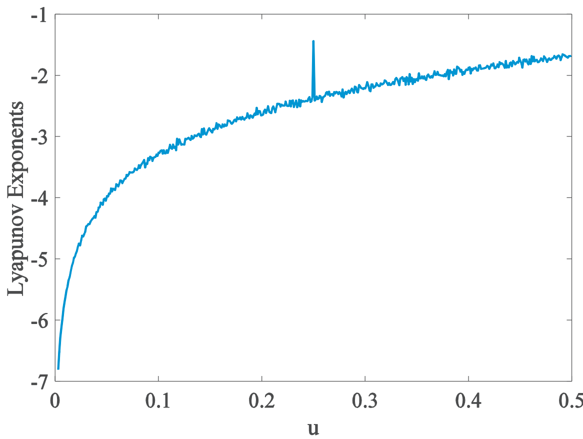

Fig. 1. Lyapunov exponents of PWLCM for different control parameter u ($u \in (0, 0.5)$).

2. Preliminaries

In this section, we briefly depict the one-dimensional piecewise linear chaotic map and the four-dimensional one-way coupled map lattice used in our hash algorithm, respectively.

2.2. Piecewise linear chaotic map (PWLCM)

We select the one-dimensional piecewise linear chaotic map in the proposed hash algorithm, which is expressed as:

$$x_{i+1} = \mathrm{PWLCM}(u, x_i) = \begin{cases} x_i/u 0 \le x_i < u \\ (x_i - u)/(0.5 - u) u \le x_i < 0.5 \\ (1 - u - x_i)/(0.5 - u) 0.5 \le x_i < 1 - u \\ (1 - x_i)/u 1 - u \le x_i \le 1 \end{cases} \quad (1)$$

where x_i represents the iteration trajectory value, and u denotes the control parameter. When u is assigned values in (0,0.5), x_i evolves into a chaotic state in range of (0,1). The PWLCM has properties of uniform distribution, good ergodicity, confusion and diffusion, therefore, it can provide chaotic random sequences. To show the quality of PWLCM, we plot the Lyapunov exponents and the bifurcation diagram for different values of control parameter u in Figs. 1 and 2. As illustrated in Figs. 1 and 2, for $0 < u < 0.5$, PWLCM can exhibit chaotic behavior.

2.3. One-way coupled map lattice (OCML)

We select the four-dimensional one-way coupled map lattice in the proposed hash algorithm, which is expressed as:

$$\begin{bmatrix} x_1(i+1) \\ x_2(i+1) \\ x_3(i+1) \\ x_4(i+1) \end{bmatrix} = OCML(e, x_1(i), x_2(i), x_3(i), x_4(i))$$

$$= \begin{cases} x_1(i+1) = (1-e)g(x_1(i)) + eg(x_4(i)) \\ x_2(i+1) = (1-e)g(x_2(i)) + eg(x_1(i)) \\ x_3(i+1) = (1-e)g(x_3(i)) + eg(x_2(i)) \\ x_4(i+1) = (1-e)g(x_4(i)) + eg(x_3(i)) \end{cases}$$

where $g(x) = 4x(1\text{-}x)$, and e denotes a coupling constant in the range of (0,1). The four inputs $x_1(i)$, $x_2(i)$, $x_3(i)$, and $x_4(i)$ are in the range of [0,1], and the corresponding four outputs $x_1(i+1)$, $x_2(i+1)$, $x_3(i+1)$, and $x_4(i+1)$ belong to the range of [0,1] as well [34].

3. Description of hash algorithm

In this section, we design the circular shift based chaotic hash function with variable parameters, which is composed of parameter initialization, message processing and hash value generation. In addition, we describe the proposed algorithm in Algorithm 1 and illustrate the structure of the proposed hash function in Figs. 3 and 4. As designed, the input is an arbitrary length of message M' and the output is $h = 128$ bits of hash value.

3.1. Parameter initialization

We first convert the original message M' into ASCII code values based on the ASCII code chart and then save them in an array M, where the length of M is denoted as l. Then, with initial value $x_0 =$ 0.8and parameter $u_0 = 0.232323$, we iterate the PWLCM (128+l) times to obtain an array X. Then we round the first 128 elements of X to the nearest integers, and assign them to the initial hash value H_0 ($H_0 = (X[1], X[2], ..., X[128])$), where $H(i)$ ($i = 1, 2,..., 128$) are binary values. Finally, we assign the rest elements of the array X to a new array U ($U = (X[129], X[130], ..., X[128 + l])$), where $U(i)$ ($i = 1, 2,..., l$) are pure decimals.

3.2. Message processing

Since each element (character) of array M is processed in the same mode, we choose $M(n)$ ($n = 1,2,\ldots,l$) as a representative to illustrate the generation process of the nth middle hash value H_n. For element $M(n)$, we iterate PWLCM $i = 132$ times to generate an array uX with the initial value $ux_0 = \frac{M(n)}{256}$, and variable parameters $uu_n = \frac{uu_0 + ux_{n-1}}{3}$ ($i = 1, 2, ..., 132$), where $uu_0 = |w - \lfloor w \rfloor - 0.5|$ and $w = \frac{(n+1)\times(M(n)+2)}{(n+2)\times(M(n)+1)} + U(n)$. Then, we round the first 128 elements of uX into the nearest integers, and assign them to the four initial buffers A, B, C, D as: $A = (uX(1), uX(2), ..., uX(32))$,

$$B = (uX(33), uX(34), ..., uX(64)),$$

$$C = (uX(65), uX(66), ..., uX(96)),$$

$$D = (uX(97), uX(98), ..., uX(128)).$$

We assign the rest four elements of uX as:$x_1 = uX(129)$, $x_2 = uX(130)$, $x_3 = uX(131)$, and $x_4 = uX(132)$, which are the input parameters of OCML. With above parameters, we iterate $OCML(uX(i), x_1, x_2, x_3, x_4)$ $i = 128$ times and we obtain four sequence values of x_1, x_2, x_3, x_4, each with length of 128. Then, we assign values as: $ch_1(i) = \mathrm{mod}(\lfloor x_1(i) \rfloor \times 1000, 32) + 1$; $cl_1(i) = \mathrm{mod}(\lfloor x_1(i) \rfloor \times 10000, 32)$; $ch_2(i) = \mathrm{mod}(\lfloor x_2(32+i) \rfloor \times 1000, 32) + 1$; $cl_2(i) = \mathrm{mod}(\lfloor x_2(32+i) \rfloor \times 10000, 32)$; $ch_3(i) = \mathrm{mod}(\lfloor x_3(64+i) \rfloor \times 1000, 32) + 1$; $cl_3(i) = \mathrm{mod}(\lfloor x_3(64+i) \rfloor \times 10000, 32)$; $ch_4(i) = \mathrm{mod}(\lfloor x_4(96+i) \rfloor \times 1000, 32) + 1$; $cl_4(i) = \mathrm{mod}(\lfloor x_4(96+i) \rfloor \times 10000, 32)$.

For buffers A, B, C, and D, we conduct i-time ($i = 1,2,\ldots,32$) value switching and left circular shifts as: swap $A(i)$ with $A(ch_1(i))$, and then apply $cl_1(i)$-bit left circular shifts; swap $B(i)$ with $B(ch_2(i))$, and then apply $cl_2(i)$-bit left circular shifts; swap $C(i)$ with $C(ch_3(i))$, and then apply $cl_3(i)$-bit left circular shifts; swap $D(i)$ with $D(ch_4(i))$, and then apply $cl_4(i)$-bit left circular shifts. Then, we obtain updated buffer values A, B, C, and D, which are then cascaded to generate the middle hash value H_n. Meanwhile, we store values as: $chf_1(n) = ch_1(1) + ch_2(1) + ch_3(1) + ch_4(1)$, $chf_2(n) = ch_1(32) + ch_2(32) + ch_3(32) + ch_4(32)$, and $clf(n) = cl_1(32) + cl_2(32) + cl_3(32) + cl_4(32)$.

Download English Version:

https://daneshyari.com/en/article/8254697

Download Persian Version:

https://daneshyari.com/article/8254697

Daneshyari.com