A Non-chaining Message Authentication Code Based on Chaos

Di Xiao, Qingqing Fu, and Tao Xiang

College of Computer Science, Chongqing University, Chongqing 400044, China
Email: xiaodi_cqu@hotmail.com, 806745280@qq.com, txiang@cqu.edu.cn

Abstract–It may be difficult for traditional chaining type Message Authentication Code (MAC) to work efficiently in parallel computing environment. In this paper, a chaos-based non-chaining MAC for parallel realization is proposed, whose structure can ensure the uniform sensitivity of MAC value to the message. By means of the mechanism of both changeable-parameter and self-synchronization, the keystream establishes a close relation with the algorithm key, the content and the order of each message block. The entire message is modulated into the chaotic iteration orbit, and the coarse-graining trajectory is extracted as the MAC value. Theoretical analysis and computer simulation indicate that the proposed algorithm can satisfy the performance requirements of MAC. It is a good choice for MAC on parallel computing platform.

1. Introduction

Message Authentication Code (MAC) is a basic technique for information security [1]. Hash function is a special kind of one-way function which can be classified into the following two categories: unkeyed hash function, whose specification dictates a single input parameter, a message; and keyed hash function, whose specification dictates two distinct inputs, a message and a secret key. The keyed hash function can also be used as Message Authentication Code (MAC). Most keyed hash functions are designed as the traditional chaining structure, which is essentially realized in a sequential mode. The processing of the current message block cannot start until the previous one has been processed. This limitation restricts their applications on the platform supporting parallel processing. Besides, the sensitivities of MAC value to the message blocks at different positions of the message are uneven.

Recently, a variety of chaos-based MAC or hash functions have been proposed [2]-[4]. However, they still inherit the traditional chaining structure. Therefore, they inevitably have the above same flaws.

In this paper, a chaos-based non-chaining MAC for parallel realization is proposed, whose structure can ensure the uniform sensitivity of MAC to the message. The mechanism of both changeable-parameter and self-synchronization is utilized to achieve the performance requirements of MAC. The rest of this paper is arranged as follows. Section 2 introduces the algorithm and its characteristics in detail. In Section 3, its performance analysis is given. Section 4 is the conclusion.

2. The proposed algorithm

2.1. Algorithm Structure

Let $N = 128$ be the bit-length of hash value without loss of generality. First of all, the original message M is padded such that its length is a multiple of 128: let m be the length of the original message M; the padding bits $(100 \cdots 0)_2$ with length n (such that $(m+n) \bmod 128 = 128 - 64 = 64$, $1 \le n \le 128$) are appended. The left 64-bit is used to denote the length of the original message M. If m is greater than 2^{64}, then $m \bmod 2^{64}$. After padding, M is constituted by blocks with 128 bits, $M = (M_1, M_2, \cdots, M_s)$, and each block is indicated as $M_i = M_i^1 M_i^2 \ldots M_i^N$. The initial N-bit vector H_0 is set.

The whole structure of the algorithm can be illustrated in Fig. 1 and described in (1).

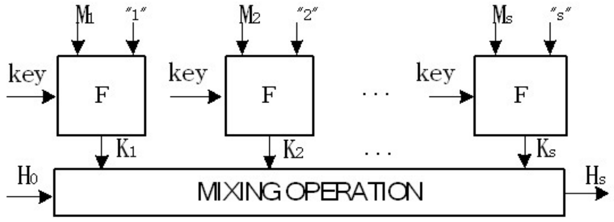

Fig.1 General model for the non- iterated type MAC

$$\begin{cases} K_i = F(key,\ i,\ M_i),\ \ i = 1,2,\cdots,s \\ MAC\ (M) = H_s = MIXING\ \ (H_0, K_1, K_2, \cdots, K_s) \end{cases} \qquad (1)$$

where M_i is the i^{th} message block; i is the order of each block; and $MAC(M)$ is the final MAC value. The whole algorithm consists of two parts: In the first part, the process of the i^{th} message block M_i, the core task can be summarized as: under the control of $(key,\ i,\ M_i)$, the keystream K_i is generated by compression function F. The above keystream K_i ($i = 1,\ 2,\ \cdots,\ ,s$) corresponding to different message block M_i ($i = 1,\ 2,\ \cdots,\ ,s$) can be generated in a parallel mode, respectively. In the second part, based on all the generated keystream K_i ($i = 1,\ 2,\ \cdots,\ ,s$), the MIXING operation are performed on $H_0, K_1, K_2, \cdots, K_s$ to get the final MAC value. Through the analysis on (1), we may notice that the effect of each message block M_i on the final MAC value $MAC\ (M)$

is equivalent. This overcomes the flaw in existing algorithms that the sensitivities of MAC value to the message blocks at different positions of the message are wavy.

2.2. Algorithm Description

Piecewise Linear Chaotic Map （PWLCM） is:

$$x_{(k+1)} = F_u(x_k) = \begin{cases} x_k/u, & 0 \le x_k < u \\ (x_k - u)/(0.5 - u), & u \le x_k < 0.5 \\ (1 - x_k - u)/(0.5 - u), & 0.5 \le x_k < 1 - u \\ (1 - x_k)/u, & 1 - u \le x_k \le 1 \end{cases} \quad (2)$$

where $x_k \in [0,\ 1]$, $u \in (0,\ 0.5)$ are the iteration trajectory value and parameter of PWLCM, respectively. The initial x_0 and u_0 are set as the algorithm key.

The i^{th} 128-bit message block M_i ($i = 1,\ 2,\ \cdots,\ ,s$) is re-divided into L units as (3), each unit has 8 bits (actually a character).

$$M_i = \underbrace{M_i^1 M_i^2 \cdots M_i^8}_{w_0}, \underbrace{M_i^9 M_i^{10} \cdots M_i^{16}}_{w_1}, \cdots \underbrace{M_i^{N-7} M_i^{N-6} \cdots M_i^N}_{w_L} \quad (3)$$

Step 1. The current order “ i ” is normalized as $z_i \in [0,\ 1]$ by computing $z_i = (i-1)/(s-1)$, and $uu_0 = (u_0 + z_i)/2 \in (0,\ 1)$ is set. Then, $w_1, w_2, \cdots, w_L$ of M_i is pre-mapped into $wr_1, wr_2, \cdots, wr_L$ in $[0,\ 1)$ by means of linear transform $wr_i = w_i / 256,\ i = 1,2,\cdots,L$.

Step 2. The iteration process of PWLCM is as follows:

(a) $j = 1$: Set the initial condition and parameter $y_0 = x_0$, $P_1 = (wr_1 + uu_0)/4 \in (0,0.5)$, respectively, and iterate 8-time F_{P_1} , then obtain $\{y_r\}, r = 1,2,\cdots,8$;

(b) $j = 2 \sim L$: Set the initial condition and parameter $y_0 = y_{8\times(j-1)}, P_j = (wr_j + y_{8\times(j-1)})/4 \in (0,0.5)$,respectively, and iterate 8-time F_{P_j} , then obtain $\{y_r\}, r = 8\times(j-1)+1,\ 8\times(j-1)+2,\cdots,8(j-1)+8$;

(c) $j = L+1$: Set the initial condition and parameter $y_0 = y_{8\times L}, P_j = (wr_L + y_{8\times L})/4 \in (0,0.5)$, respectively, and iterate 8-time F_{P_j} , then obtain $\{y_r\}, r = 8\times L+1,\ 8\times L+2,\cdots,\ 8\times L+8$;

(d) $j = L+2 \sim 2\times L$: Set the initial condition and parameter $y_0 = y_{8\times(j-1)}, P_j = (wr_{(2\times L-j+1)} + y_{8\times(j-1)})/4 \in (0,0.5)$, respectively, and iterate 8-time F_{P_j} , then obtain $\{y_r\}, r = 8\times(j-1)+1,\ 8\times(j-1)+2,\cdots,\ 8\times(j-1)+8$.

Step 3. Obtain $\{yy_r\}$, $r = 1,\ 2,\ \cdots,N$ from $\{y_r\}$, $r = 1,\ 2,\ \cdots,2N$ by setting $yy(j) = y(j+N)$, $j = 1,\ 2,\cdots,N$.

Step 4. Utilize the same method in [5] to extract all the 2nd bits of $\{yy_r\}$, $r = 1,\ 2,\ \cdots,N$ and form a binary sequence, which is composed of independent and identically distributed (i.i.d.) binary random variables.

Let x represent yy_r, $r = 1,\ 2,\ \cdots,N$, respectively. Denote the a floating point number x as

$$x = 0.b_1(x)b_2(x)\cdots b_i(x)\cdots;\quad x \in [0,1], b_i(x) \in \{0,1\} \quad (4)$$

The i^{th}-bit $b_i(x)$ can be expressed as

$$b_i(x) = \sum_{q=1}^{2^i-1} (-1)^{q-1} \Theta_{(q/2^i)}(x) \quad (5)$$

where $\Theta_t(x)$ is a threshold function which is defined as

$$\Theta_t(x) = \begin{cases} 0 & x < t \\ 1 & x \ge t \end{cases} \quad (6)$$

Set $i = 2$, a binary sequence $B_2^r = \{b_2(yy_r)\}_{r=1}^N$ (where r is the length of the sequence and yy_r is the rth floating point value) can be obtained. This N -bit sequence is K_i , the keystream corresponding to ith message block M_i, generated by compression function F .

The keystream K_i ($i = 1,\ 2,\ \cdots,\ ,s$) corresponding to different message block M_i ($i = 1,\ 2,\ \cdots,\ ,s$) can be generated in a parallel mode, respectively, and then XOR operations are performed all together to obtain the final $MAC(M)$. The most important point is that the generation of the keystream K_i must be under the control of the corresponding $(key,\ M_i,\ i)$, namely, K_i must have a close relation with the algorithm key, the content and the order of current message block M_i, which can guarantee the security.

During the process of each message block M_i ($i = 1,\ 2,\ \cdots,\ ,s$), cipher block chaining mode (CBC) [1] is introduced to ensure that the parameter P in each iteration is dynamically decided by the last-time iteration value and the corresponding message bit in different positions. On the one hand, perturbation is introduced in a simple way to avoid the dynamical degradation of chaos; on the other hand, self-synchronizing stream is realized [1], which ensures that the generated keystream K_i is closely related to the algorithm key, the content and the order “ i ”of each message block

$M_i (i = 1, 2, \cdots, , s)$. Different message block M_i leads to different keystream K_i ; and even the same message block M_i , when the order " i " changes, the corresponding keystream K_i will be also totally different.

3. Performance analysis

3.1. Distribution of MAC Value

Uniform distribution is one of the most important requirements of MAC value, which is directly related to security. Simulation experiment has been done on the following message-"*As a ubiquitous phenomenon in nature, chaos is a kind of deterministic random-like process generated by nonlinear dynamic systems. The properties of chaotic cryptography includes: sensitivity to tiny changes in initial conditions and parameters, random-like behavior, unstable periodic orbits with long periods and desired diffusion and confusion properties, etc. Furthermore, benefiting from the deterministic property, the chaotic system is easy to be simulated on the computer. Unique merits of chaos bring much promise of application in the information security field.*"

2-dimensional graphs are used to demonstrate the differences between the original message and the final MAC value. In Fig.2, the ASCII codes of the original message are localized within a small area; while in Fig.3, the hexadecimal MAC value spreads around very uniformly. No information (including the statistic information) of the original message can be left after the diffusion and confusion.

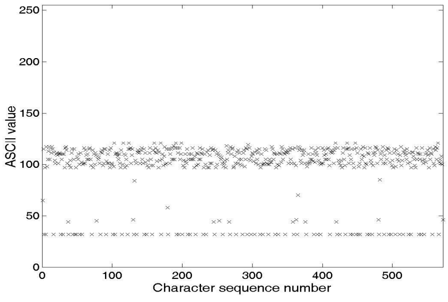

Fig.2 Distribution of the original message in ASCII

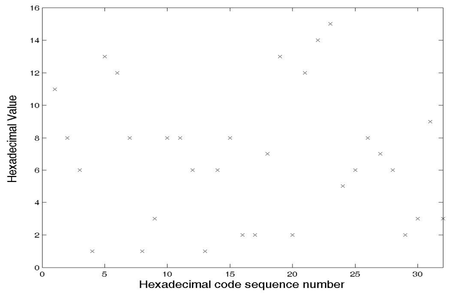

Fig.3 Distribution of the MAC value in hexadecimal format

3.2. Sensitivity of MAC Value to the Message and Key

In order to evaluate the sensitivity of MAC value to the message and secret key, MAC simulation experiments have been done under the following different 8 conditions:

C1: The original message is the same as the one in Section 3.1;

C2: Changes the first character A in the original message into B;

C3: Changes the word unstable in the original message into anstable;

C4: Changes the full stop at the end of the original message into comma;

C5: Adds a blank space to the end of the original message;

C6: Changes the secret key x_0 from 0.232323 to 0.2323230000000001;

C7: Changes the secret key u_0 from 0.454445 to 0.454445000000001;

C8: Exchanges the 1st message block M_1 -"*As a ubiquitous* " with the 2nd message block M_2 - "*phenomenon in na*".

The corresponding MAC values in hexadecimal format are gotten as follows:

C1: B861DC813886168227D2CEF568762393
C2: CF5BEEA2754AD50F9F86C66AD1F758B2
C3: B2777A4C86BA12A6217018D50983284A
C4: 58839676B70448E52B34A38E472E1B23
C5: BD0C93DCD8C87A3D55B896065CB67B7E
C6: 9854FDF54E412544FBA73676EADBD580
C7: 5C3CCDFE32CBA412BA21DD1ECE3DA7A5
C8：F504F69E60E1CD399945A1D80ED1816F.

The simulation result indicates that the sensitivity property of the proposed algorithm is so perfect that any least difference of the message or key will cause huge changes in the final MAC value. The initial condition x_0 and initial parameters u_0 of PWLCM are set as the algorithm secret key. The key space is large enough to resist all kinds of brute-force attacks. Moreover, for the sensitivity to tiny changes in initial conditions and parameters of chaotic map, it is impossible to deduce x_0, u_0 from the iteration value.

3.3. Statistic Analysis of Diffusion and Confusion

For the MAC value in binary format, each bit is only 1 or 0. So the ideal effect should be that any tiny changes in original conditions lead to the 50% changing probability for each bit of MAC value. We have performed the following diffusion and confusion test: A paragraph of message is randomly chosen and MAC value is generated; then a bit in the message is randomly selected and toggled and a new MAC value is generated. Two MAC values are compared and the number of changed bit is counted as B_i .

This kind of test is performed N -time. Four statistics are defined:

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

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

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

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

Through the tests with N =256, 512, 1024, 2048, respectively, the corresponding data are listed in Table I. 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 indicates the capability for diffusion and confusion is stable.

Table 1 Statistics of the number of changed bit B_i

	N=256	N=512	N=1024	N=2048	Mean
$\overline{B}$	63.9023	63.7715	64.0742	63.9673	63.9288
P/%	49.92	49.82	50.06	49.97	49.9425
ΔB	5.7205	5.5125	5.6750	5.7637	5.6679
ΔP/%	4.47	4.31	4.43	4.50	4.4275

3.4. Analysis of Collision Resistance

The following test has been performed [2]-[4]: first, the MAC 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 toggled. A new MAC value is then generated and stored in ASCII format. Two MAC values are compared, and the number of ASCII character with the same value at the same location in the MAC value, namely the number of hits, is counted. Moreover, the absolute difference of two MAC values is calculated using the formula: $d = \sum_{i=1}^{N} | t(e_i) - t(e_i^{'}) |$, where e_i and $e_i^{'}$ be the i^{th} ASCII character of the original and the new MAC value, respectively, and the function $t(*)$ converts the entries to their equivalent decimal values. This kind of test has been performed 2048 times, with the key $x_0 = 0.232332, u_0 = 0.454445$. The maximum, mean, minimum values of d and Mean/character are listed in Table II. The distribution of the number of hits is given in Fig.4. It is noticed that the maximum number of equal character is only 2 and the collision is very low.

Table 2 Absolute differences of two hash values

Maximum	Minimum	Mean	Mean/character
2224	653	1466.3	91.644

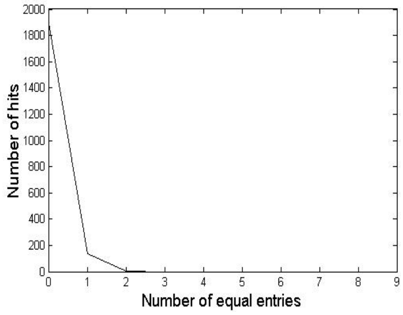

Fig.4 Distribution of the number of ASCII characters with the same value at the same location in the MAC value

3.5. Analysis of Efficiency

Since the proposed algorithm can support parallel mode, therefore its efficiency is predominant compared to other hash algorithms in sequential mode.

4. Conclusion

In this paper, a chaos-based non-chaining MAC for parallel realization, whose structure can overcomes the existing flaw that the sensitivities of MAC value to the message blocks at different positions of the message are wavy, is proposed. The proposed algorithm fulfils the performance requirements of MAC, which is with high potential to be adopted for e-Business.

Acknowledgments

The work described in this paper was funded by the National Natural Science Foundation of China (Grant Nos. 61173178, 61272043, 61472464) and the Fundamental Research Funds for the Central Universities (Grant Nos. 106112013CDJZR180005, 106112014CDJZR185501).

References

[1] B. Schneier, Applied Cryptography, Wiley, New York, 1996.

[2] D. Xiao, X.F. Liao, and K.W. Wong, “Improving the security of a dynamic look-up table based chaotic cryptosystem”, IEEE Trans. Circuits Syst. II, vol. 53, no. 6, Jun. 2006, pp. 502-506.

[3]G. Arumugam, V. Lakshmi Praba, S. Radhakrishnan, “Study of chaos functions for their suitability in generating Message Authentication Codes”, Appl. Soft Comput. vol. 7, no.3, Jun. 2007, pp. 1064-1071.

[4] A. Kanso, M. Ghebleh, “A fast and efficient chaos-based keyed hash function”, Commun. Nonlinear Sci. Numer. Simul., vol. 18, no.1, Jan. 2013, pp.109–123.

[5] T. Kohda, A. Tsuneda, “Statistics of chaotic binary sequences”, IEEE Trans. Inform. Theory, vol. 43, no. 1, Jan. 1997, pp. 104-112.