Hardware Implementation of Cryptographic Hash Function Based on Spatiotemporal Chaos

Yuling Luo, Junxiu Liu(✉), Lvchen Cao, Jinjie Bi, and Senhui Qiu

Faculty of Electronic Engieering,
Guangxi Normal University, Guilin 541004, Guangxi, China
liujunxiu@gxnu.edu.cn

Abstract. A hardware implementation of novel hash generator, namely LDHG, is proposed in this paper which is based on a spatiotemporal chaos algorithm. The proposed hash generator includes a spatiotemporal chaos algorithm computing module, message input/output port, data cache and hash code generation module. The hardware design process, security and performance evaluation are presented. Using the message authorization in smart grid as an application example, experimental results show that the proposed hash generator is irreversible, sensitive to the message and chaos parameters. It can efficiently defend the attack of invasion and forgery and the hardware area overhead is relatively low.

Keywords: Spatiotemporal chaos · Hash function · Hardware implementation

1 Introduction

Information security becomes increasing challenge due to the large scale attacks. Cryptographic hash function has been used in various information security field, e.g. digital signatures, message authentications and data corruption detections [1], which takes an arbitrary block of message and outputs a fixed-size value. The typical hash function, namely MD4 algorithm, was proposed in [2]; after that, various message-digest algorithms have been proposed, e.g. MD5 [3], SHA-0, SHA-1, PIPEMD-160, Whirlpool [4] and Whirlwind [5]. However, due to the fact that they are constructed by arithmetical operations or multi-round iterations of ciphers, their security have been compromised under various attacks (e.g. the proposed collisions attacks in the approaches of [6, 7]). Therefore, researchers have looked to develop more secure and efficient hash functions. Because chaos has cryptography characteristics (i.e. random-like and ergodicity) and is extremely sensitive to initial conditions and system parameters, chaos theory has been employed to construct the hash functions.

Recently, various hash functions using chaotic maps, chaotic neural networks and parallel construction methods have been proposed. For example, a generalized Henon-based hash function was constructed in [8]; one-way hash functions based on hyper-chaotic cellular neural network and unified chaotic system were proposed in [9] and [10] respectively. Although these approaches improve the system security, however, the algorithms are not efficient due to the serial computing [11]. In order to overcome this weakness, a parallel hash function construction based on chaotic neural

J. Tian et al. (Eds.): SecureComm 2014, Part I, LNICST 152, pp. 395–404, 2015.
DOI: 10.1007/978-3-319-23829-6_27

network was proposed in [11]. Another approach of [12] analyzed the security performance of [11] and found that it is compromised by weak keys and forgery attacks; therefore proposed a method to improve its security. Based on the outcomes of [11, 12], some novel parallel hash functions were also proposed, e.g. [13] and [14] are based on spatiotemporal chaotic system and chaotic neural network.

The traditional chaotic systems are designed based on analogue circuits [15]. However, analogue circuits have the weaknesses of easily system parameters mismatch. Therefore the approaches of [16, 17] and [18] use digital systems to design the chaotic systems. This paper will also employ the digital system design method for the chaotic system implementation. In our previous work of [19], a parallel hash construction method, namely LD scheme, was proposed which is based on spatiotemporal chaos. In this paper, the LD scheme will be employed to implement a hash generator (i.e. LDHG) for message authorization. The LD scheme and message authorization process are outlined in Sect. 2. Section 3 presents the hardware implementation of the proposed hash generator and experimental results of a case study using in the smart grid will be given in Sect. 4. Section 5 provides the conclusion and highlights the future work.

2 LD Scheme

2.1 Spatiotemporal Chaos System

LD scheme [19] uses spatiotemporal chaos as the compress function for hash codes generation. The spatiotemporal chaos function is given by (1), where the parameter $\varepsilon \in (0,1)$ and $\mu \in (3.5699456, 4)$. LD scheme is a system with discrete-time and discrete-space but its state value is continous which is in the range of (0,1). Its output is distributed in all the space; therefore it is suitable for cryptography.

$$\begin{cases} x_{n+1}(i) = (1-\varepsilon)f(x_n(i)) + 0.5\varepsilon[f(x_n(i-1)) + f(x_n(i+1))] \\ f(x_n(i)) = \mu x_n(i)(1-x_n(i)) \end{cases} \tag{1}$$

2.2 One-Way Hash Function and Contrunction Method

The basic purpose of one-way hash function is to compress an input message string with an arbitrary length into a hash value with a fixed length, and its mathematical expression is $H = H(M) = \sum_i h(M_i)$, where h denotes a compression function, M_i is the corresponding block messages. Σ usually denotes a nonlinear combination. The message can be divided to several blocks where the message length in each block is determined by compression function h. Then the separated message blocks can be processed in parallel to get their respective output values. Finally all the values are mixed to obtain a final hash value. As the spatiotemporal chaotic system has a stronger 2D spatiotemporal complexity and mixture, therefore it is suitable for constructing a hash function.

3 Hardware Implementation of LDHG

Based on the construction method of hash function, the hardware implementation of LDHG is presented in this section. The implementation procedure is presented in detail and a case study in the smart grid is also given.

3.1 Hash Coding System Structure

The LDHG system structure is presented in Fig. 1. It receives the message from a *message input* port and outputs the hash code via *hash code output* port. The message input is the communication interface which is connected to other processors (such as mirco-processors). The LDHG system includes the following modules - data preprocessing, packet check, FIFO, hash code output and spatiotemporal chaos computing (including two sub-modules, data computing core and single module computing module). The functions of the first three modules are presented as follows: (a) ***the data preprocessing module*** converts the received packets (byte stream of 8-bit width) to a uniform format. Because the communication protocol varies, the packets of different protocols are converted to the standard format for the following modules processing; (b) ***the packet check module*** checks the received packet and outputs the packet length. The packet length varies for different packet; however, the coupled map lattice length is constant. Therefore, if the packets length is shorter than the length of lattice then the packet will be filled to be the full length. The output packets are forwarded to the next module for processing and (c) ***the FIFO module*** saves the received packets temporarily. The output of FIFO is connected to the spatiotemporal chaos computing module and triggers it to start the chaos function computing to generate the chaos sequences. The *data preprocessing*, *packet check* and *FIFO* modules complete the packets format, check and temporary save, and then send the standard packets sequentially to the next module – spatiotemporal chaos computing module.

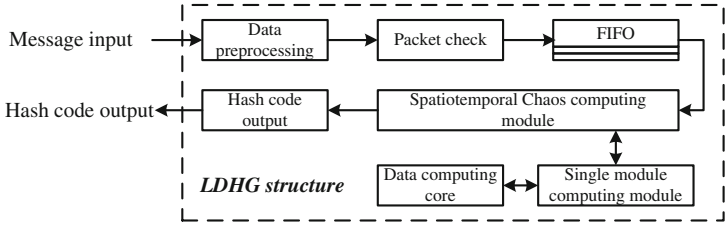

Fig. 1. Hash coding system structure

After received the packets, ***the spatiotemporal chaos computing module*** judges the length of packets, distributes the packets to different message blocks and then send the packets to the chaos system where every state value will be generated. After several times of interactions, the state value will be obtained and processed in order to generate the hash code. The hash codes of all the message blocks will be calclulated together to

generate the final hash code. The chaos computing of multiple blocks can be executed in a parallel or serial manner. For the parallel computing procedure, after all the received block packets are ready, they are sent to the computing modules simultaneously to generate the hash codes. The separated hash codes will be further calculated to get the final hash code. However, for the serial computing procedure, the hash code of each block is calculated sequentially and the final hash code is generated using the previous results of separated blocks. The main advantage of parallel computing is the high speed computing however its area overhead is a little high. The main advantage of serial computing is that the area overhead is relative low, but its computing speed is slow. The strategy selection can be made based on application field. However, no matter using parallel or serial computing procedure, for the message hash code generation, the main function is the computing in a single block. Therefore, the single block computing unit is the core module of the LDHG system.

The single block computing unit receives one block packets and generates the corresponding hash code. The input packet data (8-bit width integer) is scaled to the initial value (float point) of chaos function. After calculation of n times iteration, the final hash code (16 bytes) is generated, which is a combined value of the final chaos states. Based on the spatiotemporal chaos equation in (1), a large storage space is required if n and i is great. For example, if $n = 1000$, $i = 64$ and the data type is single float, the required space is 1000*64*32 bits = $2.048*10^6$ bits. It introduces a significant hardware cost. However, it should be noted that the calculation of $x_{n+1}(i)$ is only related to $x_n(i-1)$, $x_n(i)$ and $x_n(i+1)$. Therefore, the storage space can be minimized to 2 lines only, i.e. 2*64*32 bits = 4096 bits, which decreases the storage space efficiently. Therefore, the required storage space is *2*i*data_width*, where i is coupled map lattice and *data_width* is the width of corresponding data type.

The working flow of the single block computing unit is presented in Fig. 2. The initial state of the unit is 'S1. Idle'. If the data present is valid, the state will be changed to 'S2. Read the data to line #0' where the initial parameters of chaos function are set. Then the state changes to 'S3. Calculate Line #1' where the data in line #1 are generated based on the value of line #0. After finished calculation, the state will be changed to 'S4. Move line #1 to #0' where the data in line #1 will be moved to line #0 to be the initial value of next round calculation. If the calculation of total n round is

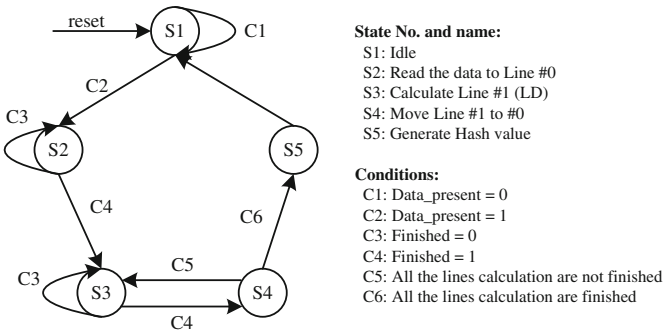

Fig. 2. The single block computing unit procedure

completed, the state will be changed to 'S5. Generate Hash value'; then the hash code of this block packet is generated. During this process, the chaos calculation in the state S4 is completed by core computing module.

The core computing module receives the data of $x_n(i-1)$, $x_n(i)$ and $x_n(i+1)$ and outputs the $x_{n+1}(i)$, see (1). In order to use a simplified chaos function equation, equation (1) can be simplified to $x_{n+1}(i) = 3.591x_n(i) - 3.591x_n^2(i) + 0.1995x_n(i-1) - 0.1995x_n^2(i-1) + 0.1995x_n(i+1) - 0.1995x_n^2(i+1)$, where $\varepsilon = 0.1, \mu = 3.99$. A parallel computing structure in Fig. 3 is used to process the input data of $x_n(i-1)$, $x_n(i)$ and $x_n(i+1)$ simultaneously. The calculation is divided to 5 stages. In the stage 1, six multipliers calculate simultaneously which can complete the multiplication operation of six pairs of decimals. Stage 2 and 3 complete the multiplication and subtraction operation of three pairs of decimals. The results are generated after calculating in the addition operation of stage 4 and 5. However, it should be noted that not all the operations run simultaneously. For example, the input of $x_n(i-1)$ has two multipliers in the stage 1 and one multiplier in stage 2. However, the input of multiplier in stage 2 is the outputs of stage 1. Therefore, the multiplier of stage 2 can use the multiplier of stage 1 in a time division multiplexing manner. Similarly, the adders and subtracters of stage 3, 4 and 5 (i.e. A3, A4, A5) can use the same physical adder. Note that the physical adder and substracter can be switched by an input signal control. Therefore, the required hardware resources is decreased from 9 multipliers, 3 adders and 2 subtracters to 6 multipliers, 3 adders which reduces the hardware area overhead efficiently.

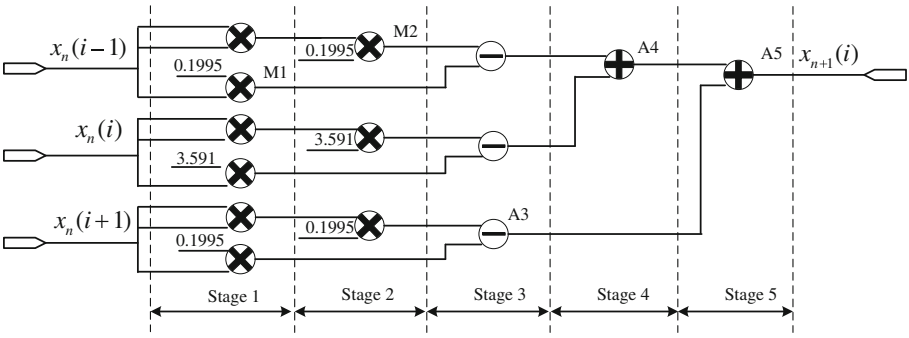

Fig. 3. The core computing unit structure

4 Performance Analysis

In this section, the LDHG is evaluated in a case study of the smart grid application. The LDHG can be used for the data encryption/decryption and the message authentication between the server and data collector. The data collector is the communication bridge between the server and the power meters. It collects the data from the power meters and forwards the data to the server. The communication between the server and data collector needs to be secure; therefore the message authentication is used to

guarantee the communication be legal. The main processor of the data collector is an ARM microcontroller. The software works under the Linux 2.6.30 operation system. The data collector can run for a long time reliably and communicates with other devices promptly. The performance analysis of the LDHG is based on the smart grid application and the data collector platform.

4.1 Security Performance Analysis

(1). Plaintext sensitivity analysis: If a hash function algorithm can generate the hash values which changes more than 50 % when the plaintext has a small change, then this algorithm is sensitive to the plaintext. In order to evaluate the plaintext sensitivity of the proposed scheme, a data set is used for the testing - C1 is the original message and C2-C6 is the modified message which only has small change. The message of C1 is a real data using in the smart grid which is the communication data of 'server reads the power meter directly'.

C1 (original): 0x68, 0xa6, 0x00, 0xa6, 0x00, 0x68, 0x4b, 0x01, 0x44, 0x01, 0x00, 0x8a, 0x10, 0x68, 0x00, 0x00, 0x01, 0x01, 0x1f, 0x00, 0x20, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x10, 0x90, 0x00, 0x00, 0x16; ***C2***: the 1st byte of C1 is changed from 0x68 to 0x69; ***C3***: the 5th byte of C1 is changed from 0x00 to 0x01; ***C4***: the 11th byte of C1 is changed from 0x00 to 0x01; ***C5***: the 21st byte of C1 is changed from 0x20 to 0x21; ***C6***: the final byte of C1 is changed from 0x16 to 0x17. The message data of C1-C6 are sent to the LDHG. The corresponding hash values are - ***C1***: 617bac8899c6 9cd84d0a4245d3d96ade; ***C2***: 7959ae52f700c-ce1cda681d6302e2f4b; ***C3***: 707a6a8ef7 3416cad064be8c4d7842b4; ***C4***: f9c632 ae74dbba95165d679eb1b805aa; ***C5***: e5ba60de cd55989ec22b7416a5d36817; ***C6***: cffeca6786258baa443b8ed0274fc9b2.

It can be seen that the hash code changes hugely when the plaintext has a little change. For example, the hash code of C3 is absolutely different to the original C1 but the plaintext of C3 and C1 only have one bit difference. Therefore, the plaintext sensitivity of the proposed hash coding system is very strong which makes it suitable for the message authentication.

(2). Parameters sensitivity analysis: The hash coding system should be not only sensitive to the plaintext, but also to the chaos function parameters. In order to evaluate the proposed scheme, the experimental parameters are defined as follows. ***K1***:$\varepsilon = 0.1$ and $\mu = 3.99$; ***K2***:$\varepsilon = 0.1000001$ and $\mu = 3.99$; ***K3***:$\varepsilon = 0.1$ and $\mu = 3.9900001$. The same message data is sent to the LDHG. The corresponding hash codes are presented as follows. ***K1***: 617bac8899c69cd84d0a4245d3d96ade; ***K2***: 991b8ec75cc0a208c80ebe091bfd91f8; ***K3***: c34822e298ad70251c1bbe04172 c6364. It can be seen that the hash code has a huge change when the parameter has a slight change, e.g. ε has a slight change of 10^{-7}, i.e. from 0.1 to 0.1000001. Therefore the proposed hash coding system is also sensitive to the key.

(3). Security analysis of diffusion and confusion: Diffusion and confusion are two essential design metrics for hash functions. Hash functions requires the message

to diffuse its effects into the entire hash space, which means that the correction between message and the corresponding hash code should be as small as possible. The following statistics are used to evaluate the security of hash function. The mean changed bit number, namely $\bar{B}$, is defined as $\bar{B} = \frac{1}{N}\sum_{i=1}^{N} B_i$, where N is the number of statistics data set, B_i is the number of changed bits at time i. The mean changed probability P is defines as $P = \bar{B}/HL \times 100\,\%$, where HL is the length of hash codes. The standard variance of the changed bit number ΔB, is defined as $\Delta B = \sqrt{[1/(N-1)]\sum_{i=1}^{N}(B_i - \bar{B})^2}$ and the standard variance ΔP, is defined as $\Delta P = \sqrt{[1/(N-1)]\sum_{i=1}^{N}(B_i/HL - P)^2} \times 100\,\%$. The analysis of diffusion and confusion are performed for the change of message and chaos function parameters, respectively. The results are presented in Table 1. It can be seen that when the message has one bit changed or the chaos function parameter has a little change, $\bar{B}$ is ~ 60, P is 47 % and ΔB and ΔP are very small. The ideal values of $\bar{B}$ and P are 64 and 50 %, respectively [20]. The results in this paper are closed the ideal values which indicate that the proposed work has a good diffusion and confusion capabilities.

Table 1. Statistics of changed hash codes

	Message	Parameter	Average
$\bar{B}$	60.4	59	59.7
$P(\%)$	47.187	46.094	46.641
ΔB	7.469	4.243	5.586
$\Delta P(\%)$	5.835	3.314	4.575

4.2 Computing Speed and Area Overhead

In this section, two aspects of the proposed hash generator are analyzed – computing speed and area overhead. The experimental environment is defined as follows. The parameters of chaos function are set by $\varepsilon = 0.1$, $\mu = 3.99$. The parameter i is equal to 64 according to the message length in the smart grid application. However, for parameter n, if it is greater, then the required iteration calculation is larger, the system nonlinear dynamic behavior is more complex and the generated hash codes are more secure. The parameter n will be chosen from 20 to 200 for evaluation.

(1). The computing speed of different platforms: the experimental environment of server is set as follows – Intel Core i7-2600 3.4 GHz processor, 4 G ram, 500 G hard disk, Redhat Linux enterprise 4 operating system, Gcc 3.4.3 compiler. The arm microcontroller is Atmel AT91SAM9G45 device, 400 MHz, 256 M ram, Linux 2.6.30 operating system. The LDHG uses Altera Cyclone IV E EP4CE115F29C7 device and the system clock frequency is 100 MHz. Figure 4

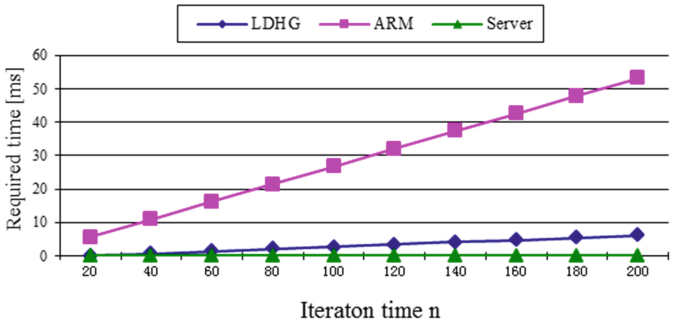

Fig. 4. The required time comparison on different processors

presents the required time of hash code generation on different platforms. It can be seen that for all the platforms the required time increases if iteration time (i.e. parameter n) increases. When n is between 20 and 200, the required time of server is between 0.013 and 0.127 ms. This time period is short and server can meet the time requirement of frequent message authorization. However, the required time of LDHG is 0.034 $\sim$ 6.124 ms and arm microcontroller is 5.267 $\sim$ 53.198 ms. According to the research outcome of [21], the chaos system is independent when $n > 35$ and is secure. In this paper, n is set to be 40; then the required time of LDHG is 0.618 ms and arm device is 10.961 ms. The latter is $\sim$16x greater than the former. And if n continues to increase, the required time of arm increases more sharply than LDHG. For example, if n = 200, the arm requires 53.198 ms which is much larger than the LDHG, i.e. 6.125 ms.

From the required time comparisons, it can be seen that the server can complete the calculation in a short time and meet the time requirement for highly frequent message. However, the calculation time of arm microcontroller is much greater than the proposed hash generator, especially when the iteration time n is large. Therefore, the proposed hash generator is more suitable than arm microcontroller when there are highly frequent message to be authorized; however lots of industrial applications have this requirement, such as smart grid.

(2). The area overhead of the proposed hash generator: for the hardware implementation of LDHG, one PLL module is used and several DSP modules, FIFO are instantiated. All the modules are design separately and connected together in the top design. The area overhead is relatively low; only 14 % logic elements (16,251/114,480 = 14 %) are used. A FPGA implementation of traditional MD5 algorithm was proposed in [22] based on a Xilinx Virtex V1000FG680 device, where the area overhead (slices) of the full-loop-unrolled is 38 % (4763/12288 = 38 %). However, the security of MD5 has been compromised [6, 7].

From the computing speed comparison of different platforms, it can be seen that the proposed LDHG has a quick computing speed, generates hash code in a short time and is suitable for frequent message authorization. In the meantime, the area overhead of hash coding module is low which is suitable for hardware implementation.

5 Conclusions

This paper proposed a hash generator based on spatiotemporal chaos systems, including the principal of chaos system, the hardware implementation and experimental results. The performances of computation speed and area overhead are evaluated. It has been applied for smart grid system and the results showed that it can complete frequent message authorization quickly and efficiently enhance the security of communication data. This paper is a beneficial exploration using nonlinear chaos system to implement a hash function and apply to the message authorization in smart grid application. The future work includes the design of chaotic cryptographic system for image and video secure transmission and data encryption mechanism.

Acknowledgements. This research was supported by the Guangxi Natural Science Foundation (2014GXNSFBA118271), the Research Project of Guangxi University of China under Grants ZD2014022 and ZD2014124, Guangxi Key Lab of Wireless Wideband Communication & Signal Processing under Grant GXKL0614205, the Education Development Foundation and the Doctoral Research Foundation of Guangxi Normal University, and the State Scholarship Fund of China Scholarship Council.

References

1. Menezes, A.J., Van Oorschot, P.C., et al.: Applied Cryptography. CRC Press, Boca Raton (1996)
2. Rivest, R.L.: The MD4 message digest algorithm. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 303–311. Springer, Heidelberg (1991)
3. Rivest, R.L.: The MD5 message-digest algorithm. In: RFC 1321 (1992)
4. Barreto, P.S.L.M., Rijmen, V.: The whirlpool hashing function. In: First open NESSIE Workshop, pp. 1–20 (2000)
5. Barreto, P., Nikov, V., Nikova, S., et al.: Whirlwind: a new cryptographic hash function. Des. Codes Crypt. **56**(2–3), 141–162 (2010)
6. Wang, X., Feng, D., Lai, X., Yu, H.: "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD", Cryptology ePrint Archive: Report, vol. 5, pp. 5–8 (2004)
7. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 1–18 (2005)
8. Zheng, F., Tian, X., Li, X., Wu, B.: Hash function based on the generalized Henon map. Chin. Phys. B **17**(5), 1685–1690 (2008)
9. Yang, Q., Gao, T.: One-way hash function based on hyper-chaotic cellular neural network. Chin. Phys. B **17**(7), 2388–2393 (2008)
10. Long, M., Peng, F., Chen, G.: Constructing a one-way hash function based on the unified chaotic system. Chin. Phys. B **17**(10), 3588–3595 (2008)
11. Xiao, D., Liao, X., Wang, Y.: Parallel keyed hash function construction based on chaotic neural network. Neurocomputing **72**(10–12), 2288–2296 (2009)
12. Wang, X., Zhao, J.: Cryptanalysis on a parallel keyed hash function based on chaotic neural network. Neurocomputing **73**(16–18), 3224–3228 (2010)

13. Li, Y., Xiao, D., Deng, S., Han, Q., Zhou, G.: Parallel hash function construction based on chaotic maps with changeable parameters. Neural Comput. Appl. **20**(8), 1305–1312 (2011)
14. Wang, Y., Wong, K.-W., Xiao, D.: Parallel hash function construction based on coupled map lattices. Commun. Nonlinear Sci. Numer. Simul. **16**(7), 2810–2821 (2011)
15. Caponetto, R., Di Mauro, A., Fortuna, L., Frasca, M.: Field programmable analog arrays to implement programmable chua's circuit. Int. J. Bifurcat. Chaos **15**(5), 1829–1836 (2005)
16. Zhou, W., Yu, S.: Design and implementation of chaotic generators based on IEEE-754 standard and field programmable gate array technology. Acta Physica Sinaca **57**(8), 4738–4747 (2008)
17. Zhou, W., Yu, S.: Chaotic digital communication system based on field programmable gate array technology - Design and implementation. Acta Physica Sinaca **58**(1), 113–119 (2009)
18. Luo, Y., Yu, S., Liu, J.: Design and implementation of image chaotic communication via FPGA embedded ethernet transmission. In: International Workshop on Chaos-Fractals Theories and Applications, pp. 148–152 (2009)
19. Luo, Y., Du, M.: One-way hash function construction based on the spatiotemporal chaotic system. Chin. Phys. B **21**(6), 060503–060510 (2012)
20. Nouri, M., Khezeli, A., Ramezani, A., Ebrahimi, A.: A dynamic chaotic hash function based upon circle chord methods. In: 6th International Symposium on Telecommunications (IST), pp. 1044–1049 (2012)
21. Yi, X.: Hash function based on chaotic tent maps. IEEE Trans. Circuits Syst. II Express Briefs **52**(6), 354–357 (2005)
22. Deepakumara, J., Heys, H.M., Venkatesan, R.: FPGA Implementation of MD5 Hash Algorithm. In: Canadian Conference on Electrical and Computer Engineering, no. 81, pp. 919–924 (2001)