A Novel Hash Function based on Hybrid Cellular Automata and Sponge Functions

Anita John[1], Ajay P Manoj[2], Alan Reji[2], Atul Premachandran[2], Basil Zachariah[2], and Jimmy Jose[1]

[1] Department of Computer Science and Engineering,
National Institute of Technology Calicut, Kozhikode, India
{anita_p170007cs, jimmy}@nitc.ac.in
[2] Department of Computer Science and Engineering,
Rajagiri School of Engineering and Technology, Kochi, India
{u1603013, u1603021, u1603065, u1603066}@rajagirtech.edu.in

Abstract. Hash functions serve as the fingerprint of a message. They also serve as an authentication mechanism in many applications. Nowadays, hash functions are widely used in blockchain technology and bitcoins. Today, most of the work concentrates on the design of lightweight hash functions which needs minimal hardware and software resources. This paper proposes a lightweight hash function which makes use of Cellular Automata (CA) and sponge functions. This hash function accepts arbitrary length message and produces fixed size hash digest. An additional property of this function is that the size of the hash digest may be adjusted based on the application because of the inherent property of varying length output of sponge function. The proposed hash function can be efficiently used in resource constraint environments in a secure and efficient manner. In addition, the function is resistant to all known generic attacks against hash functions and is also preimage resistant, second preimage resistant and collision resistant.

Keywords: Cryptographic Hash functions · Cellular Automata · Sponge Functions · Omega Flip Permutation

1 Introduction

Any digital document needs a fingerprint of itself to avoid misuse of the document known as the message digest or hash digest. Cryptographic hash functions are one-way functions that take an arbitrary length input and produce a fixed length hash digest as output. Different hash functions were developed so far which made use of Merkle-Damgård [21,10] and Davies-Meyer [20] schemes. Today, most of the applications need lightweight hash functions which can be used in resource constraint environments. But, the lightweight property of the hash functions must not affect the security of the function. So this is a trade off between efficiency and security. In order to accomplish this, new cryptographic

primitives like CA and sponge functions can be used in their design. This paper proposes a hash function that makes use of both CA and sponge function.

The rest of the paper is organized as follows. Section 2 gives a brief description about CA and sponge functions. Section 3 gives literature survey on the hash functions based on CA and sponge. Section 4 explains the detailed design of the proposed hash function followed by Section 5 that discusses the design rationale. Security analysis is done in Section 6 followed by conclusion.

2 Preliminaries

In this section, we discuss some basic definitions and terminologies related to CA, sponge functions and cryptographic hash functions.

2.1 Basics of CA

CA are a discrete lattice of cells. Each cell has a memory element and a next state computation function or rule which is a Boolean function associated with it. The value of the cells get updated based on the CA rule at every clock cycle and this happens in parallel. The transformation of the value in a particular cell is influenced by the memory elements of the neighbouring cells. The rules of CA are developed based on the number of neighbours involved in the updation of a cell. There are two types of CA rules - linear rules and non-linear rules. Linear rules are those rules that use only XOR in their combinational logic and non-linear rules are rules that do not follow the conditions of linear rules. In general, the number of cells n that participate in a CA cell update is given by $n = 2r + 1$, where r is the radius of the neighbourhood. All the cells in a CA can make use of same CA rules or different CA rules for updation. If they use the same rules, then they are called *uniform CA* and if they use more than one rule, then they are called *hybrid CA*. The neighbors of the cells at the extreme left and right are decided based on whether the CA follows periodic or null boundary conditions. In addition to these conditions, three more boundary conditions are discussed in [8] namely *one null*, *null one* and *one* boundary conditions. Here, we make use of *one null boundary* where the left most cell takes the value 1 as its left neighbour and rightmost cell takes the value 0 as it right neighbour. This makes CA rules cryptographically robust [8]. CA have recently evolved as a good cryptographic primitive due to its chaotic nature and parallel execution. It is also efficient in both hardware and software applications due to its simple logic operations. Three-neighbourhood CA developed by Stephen Wolfram [30] had been very powerful in developing cryptographically secure pseudorandom number generators. Later, 3-neighbourhood CA have been widely used in many cryptographic applications like stream ciphers [28], block ciphers [23] and hash functions [22].

2.2 Sponge Functions

Sponge function [3] is a simple iterative function that takes a variable length input, processes it and generates an infinite length output. Sponge functions have an internal function F that can either permute or transform the state bits. Sponge function is the basic technique used in Keccak[5] which was selected as the new SHA-3 standard by the National Institute of Standards and Technology (NIST) in October 2012. The function operates on a state of bits of width b, where $b = r + c$. Here r is called the *bit rate* and c is called the *capacity* of the sponge. The values of r and c can be varied. The security level that can be attained using sponge is dependent on c [4].

Working of sponge function-
Absorbing phase- In this phase, each message block after XORing with r bits of the b state bits is subject to an internal transformation function. The output of n_r rounds of transformation will be the state bits used for the next message block. This continues until all message blocks are exhausted.
Squeezing phase- In this phase, the r bits from the output of absorption phase are extracted and it forms the first part of the final hash digest. This state again undergoes transformation and the next r bits from its output are again squeezed out. This continues until the needed length hash digest has been squeezed out. If the length of the squeezed out bits is more than the required hash digest length, it must be truncated to match the needed length. The strength of the sponge function lies in the internal transformation function.

2.3 Cryptographic Hash Functions

A cryptographic hash function accepts an arbitrary length input and outputs a fixed length output. Earlier, all known hash functions made use of a compression function which processes every message block in the same manner. They were called iterated hash functions. The properties of a good hash function are (i) Collision Resistance (ii) Preimage resistance (iii) Second preimage resistance. The most popular standard hash functions were MD5 [24] and SHA [27] (Secure Hashing Algorithm) family of hash functions. SHA-256 which produces a 256-bit hash digest is considered to be secure and is now used in Blockchain. Even though MD5 and SHA were considered to be the standard hash functions, by 2004 and 2005, Wang et al [29] had shown that finding collisions for MD5 and SHA were relatively easy. This made NIST to make an announcement for an open competition in 2007 for a stronger hash function design to be selected as SHA-3. In 2012, Keccak [5] was declared as the winner. Keccak or SHA-3 used a new design strategy called ***sponge functions***. The main advantage of using sponge functions in hash design is that we could get a hash digest of any desired length.

2.4 Omega Flip Permutations

Several bit level permutation functions for fast software cryptography were discussed in [18]. Bit level permutations play an important role in introducing diffusion in a block of data. Omegaflip (OMFLIP) permutation permutes n bits in atmost $logn$ permutation instructions. The instruction used for performing this operation is *omflip*. For an omega flip network, there are 2 basic operations- omega and flip, the order of which are decided by a 2-bit opcode. In order to permute n bits, we make use of an n-bit control string. More detailed description about the working of this instruction is available in [19].

3 Related Work

3.1 Hash functions based on Cellular Automata

Ivan Damgard was the first to propose a collision-free hash function based on CA [10]. Damgard proposed a hash function that computes a 128-bit hash digest from a 256-bit string. This made use of a binary CA rule $Rule120$ with periodic boundary conditions. The CA evolved through 384 iterations and the first bits of the strings from 257^{th} to 384^{th} iterations were taken as the final hash digest. But his proposal was cryptanalyzed by Daemen [9], who in turn proposed another hash function called Cell Hash. Cell Hash made use of non-linear and linear CA rules with periodic boundary conditions which was also cryptanalyzed later.

Another hash function based on CA was proposed by Mihaljevic et al [22]. This paper proposes a dedicated one way hash function based on programmable CA. They have used an iterative hash function where the compression function and output function made use of CA. Another lightweight hash function based on CA developed by Hanin et al is LCAHASH [14]. An extension of LCAHASH is proposed in LCAHASH-1.1 [25] which made use of non-uniform CA with rule $< 30, 90 >$ for 128 or 256 iterations and the required output is taken as the hash digest. Jamil et al [16] had proposed a hash function based on hybrid CA that made use of linear and non-linear rules together in a CA along with a customized omega flip permutation. They made use of Rules 30 and 134 and they were the first to employ such elements into the compression function which showed good avalanche effect. Another recent work based on CA was done by Sadak et al in [26] where they made use of a carefully chosen cryptographically hybrid rule set which contains both linear and non-linear CA rules. The rules are selected based on Cryptographically Secure Hybrid Rules (CSHR) algorithm proposed in [8].

3.2 Hash functions based on sponge functions

A lightweight hash function based on sponge construction with permutation is Quark [2]. The hash function made use of shift registers and Boolean functions. The permutation used in QUARK was done by using stream ciphers based on shift registers like Grain [15] and block cipher like KATAN [7]. SPONGENT

[6] and PHOTON [12] were the other lightweight hash functions that made use of sponge functions in their domain extension algorithm. Keccak, the winner of NIST competition for a new hash function was also based on sponge construction. A detailed description of all lightweight cryptographic hash functions is found in B.Hammad et al in [13].

3.3 Hash functions based on CA and sponge functions

As part of the research to find lightweight and efficient cryptographic hash functions, new designs that used compression functions using CA and sponge functions were developed. The use of CA and sponge as individual primitives in these functions produced hash digests that satisfied the properties of collision resistance, preimage and second preimage resistance. The strength of a hash function that uses sponge function is the strength of the internal transformation function used in it. This inspired researchers in the field of cryptographic hash functions to use CA in the transformation function of sponge.

CASH[17] is a family of hash functions based on CA and sponge function. This hash function made use of tweakable parameters which allows the user to choose suitable parameters depending upon the level of security and efficiency required for a particular application. They made use of both permutation and transformation in the internal transformation function of the sponge function. The hash function provided diffusion and non-linearity with the help of both cryptographic primitives. Very few works have been done that made use of CA in the internal function of sponge.

4 Proposed Hash Function

This paper proposes a new hash function which makes use of CA in the internal function of sponge function. The internal transformation function has two layers- the permutation layer and the transformation layer. The transformation layer makes use of hybrid CA rules for their transformation. Here we have combined linear and non-linear 3-neighbourhood rules in a single CA which is supposed to give good characteristics to the hash function. The permutation layer makes use of Omega Flip permutation [18].

Description of the hash function

The hash function accepts an arbitrary length input message and produces 256 bit hash digest. The size of the hash digest can be varied which is an inherent property of the sponge function. The sponge function is implemented on a state bit of width b, where $b = r + c$. Here, we have taken r as 92 and c as 132. Another combination is r=92 and c=258 which gives good results and is resistant to attacks at the cost of more execution time. The block diagram for the proposed hash function is shown in Figure 1.

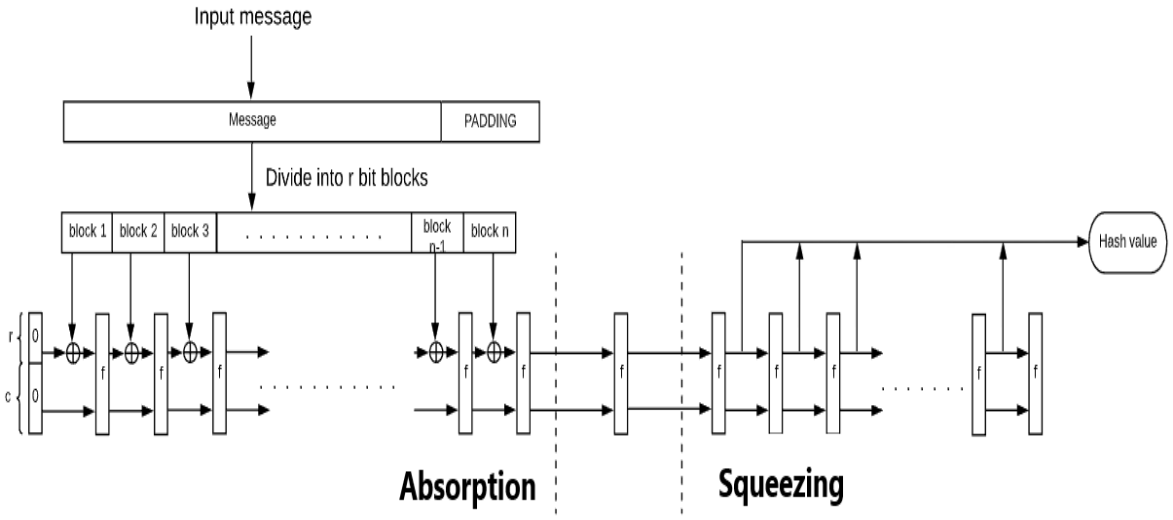

Fig. 1. Block diagram for Hash Function

The input message must be padded to make the length of the message a multiple of r. This is done by padding the message with '10' followed by the 64-bit representation of the message length. Padding step is mandatory even if the original message is a multiple of r even before padding. This ensures the collision resistance property of the hash digest. The padded message is then divided into r bit blocks M_1, M_2,..., M_i. Initially, b is set to all zeros. Each message block is XORed with the r bits of the state bits of b. The XORed r bits together with c bits will be transformed by the internal transformation function. The block diagram of the internal function F of the sponge function is shown in Figure 2.

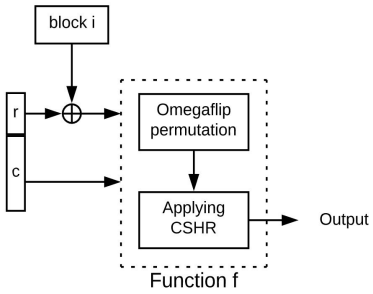

Fig. 2. Internal function

Single Round of the Internal Function
The state bits in b undergo an internal transformation function F which proceeds in 2 steps- permutation and transformation. To permute the state bits, we have used Omega Flip permutation. We have used the bits of the message block itself as the control bits for permuting them. In order to permute 224 bits, we need eight omflip instructions. The 2-bit opcode needed in each of these 8 instructions can be obtained from the first 16 bits of the b state bits. i.e., the first 16 bits of the message block itself. So, the order of performing omega or flip operation is decided by the first 16 bits of the message block M_i. The entire omegaflip permutation is dependent on the message. After performing the permutation, the state will be subjected to a transformation phase. In the transformation phase, we make use of cryptographically secure hybrid linear and non-linear 3-neighbourhood CA rules. The order of rules to be used in each of the cells is decided by an algorithm called Cryptographically Secure Hybrid Rules (CSHR) discussed in [8]. These rules are expected to give good results with respect to algebraic degree, non-linearity and period. The rule set that we use here is $< 30, 90, 150, 30, 210, 30, 90, 150 >$ where each component in the rule set represents an elementary CA rule. This rule set was proved to have good cryptographic properties like non-linearity and algebraic degree [8]. The CA is run for 13 clock cycles.

There will be two rounds (i.e., $n_r = 2$) for the internal function. After the last transformation function, a null round, i.e., a single round of the transformation function without any XOR input is performed. This marks the end of the absorption phase of the sponge function. During the squeezing phase, the r bits of the absorbing phase are squeezed out to form part of the hash digest. Now, the same transformation function used in the absorbing phase is applied and the next r bits are squeezed out. During this phase, the b bits of the state are taken as the control bits for omega flip permutation. This interleaved sequence of transformation and squeezing will continue until we get the required hash digest length. The size of the hash digest can be varied as per user needs by adjusting the number of times r bits are squeezed out. If the length exceeds the required hash digest length, the remaining bits can be truncated.

5 Design Rationale

The proposed hash function design makes use of CA and sponge function. The emergence of sponge functions as random oracle encourages their use in the design of cryptographic hash function as a replacement to the conventional Merkle-Damgård scheme. The design of the transformation function has a crucial role in the strength of the sponge function which increases the resistance of the hash function to known attacks on it. In the proposed design, the internal transformation consists of permutation and transformation. Since we combine both operations into a single function, we have combined the best possible among

them to retain the lightweight nature of the hash function. The design rationale for each of the steps in the hash function are discussed below.
1. Padding the input: Here we pad the message to make the length of the message a multiple of r, the bit rate. This padding scheme ensures that the last block never contains all zeros which is one of the criteria for using sponge functions. This padding rule also enhances the security of the hash function design.
2. Sponge functions: Merkle-Damgård scheme suffered from length extension attacks and multicollision attacks. The sponge function is known to be resistant to these attacks. This motivated us to use sponge function in our design. Our next aim was to strengthen the sponge function by designing a strong internal function. We have used a hybrid rule set that has a combination of linear and non-linear 3-neighbourhood rules. The use of hybrid CA enhances the security of the function in 2 ways- linear hybrid rules ensure diffusion of bits and non-linear hybrid rules introduce non-linearity to the function. Both non-linearity and diffusion are important factors that determine the strength of the function. We have added an additional step of permutation with the implementation of a faster method of Omegaflip [18] so as to distribute the bits throughout the state. The addition of null round in between the absorbing and squeezing phases helps to avoid sliding attacks on the sponge functions. So, we have tailored a CA with hybrid rules along with omega flip permutation into the sponge function.

6 Security and Performance Analysis

6.1 Security Analysis

1. Avalanche effect: Avalanche effect is a desirable property for hash functions. This property states that, a small change in the input should bring a great change in the output bits of the hash digest. To evaluate the Avalanche effect, we have taken a sample of 100 varying length messages. For each message, we changed the original message by one bit and calculated the Hamming distance between the hash values of the original message and the corresponding modified message. The values for Hamming distance range from 117 to 145 where the size of the hash digest is 256. This shows that even a single bit change in the input has varied around half of the output bits which proves that the proposed hash function exhibits good avalanche effect.

2. Pre-image resistance and Second pre-image resistance: The security of the proposed system is based on the internal transformation function of the sponge construction. This function is composed of hybrid CA and permutation. Given a hash digest output of size n, we need to perform 2^n operations to find a preimage or a second pre image and $2^{n/2}$ operations to find a collision. Here, the internal transformation has transformed input message block in an efficient manner. Each message block first goes through a permutation followed by CA. In CA, each bit is dependent on 27 neighbouring bits at the end of 13 CA cycles, since we are using 3-neighbourhood CA and each bit is dependent on 2q+1 neighbouring bits where q is the number of cycles of CA. The hash digest is

produced after many rounds of internal transformation and hence is dependent on all bits of the input message. So the function is resistant to pre-image, 2^{nd} pre-image and collision attacks.

3. Sliding attacks: One of the attacks on hash functions that made use of sponge function is the sliding attack [11]. Sliding attacks occur when we are able to find a *"slid pair"* of messages. In sponge functions, the first r bits of the hash digest is obtained from the output of the last internal transformation of the absorbing phase, So, there is a chance that an attacker gets some hint regarding the last block of the input message or about the internal transformation. In order to avoid this, we run an additional null round which hides the last message block.

4. Randomness Test using NIST Statistical Test Suite The randomness of hash digest is a property that makes it suitable for cryptographic applications. So to test the randomness of the hash digest output, we make use of NIST Statistical Test Suite [1]. This contains a battery of 15 tests. Each test computes the chi-square statistic of a particular parameter. This is done by comparing this parameter for the generated bit stream with the ideal value of this parameter. The ideal value will be the one that is obtained from the theoretical results of such an identical sequence of bits. This chi-square value is converted to a random probability value called P-value.
The tests in the test suite, in general, checks for the proportion of zeros and ones in the entire bit stream as well as in subblocks of the stream, checks whether there are any repeating patterns, non-periodic patterns or too many occurrences of ones in the bitstream. Based on these analyses, an output file will be generated by the test suite with relevant intermediate values such as test statistics and P-values for each statistical test. The P-values generated will help to analyze the randomness properties of the newly generated sequence [1]. The randomness test of the generated hash digests using NIST test suite affirms the cryptographic strength of the hash function against attacks.

To execute the test suite, the hash function has been used as a pseudorandom number generator to create a data stream of 10 Mb. The stream is generated by applying the hash function on an initial seed, S which is incremented by 1. We calculate H(S), H(S+1) and so on until we are able to concatenate them to get a 10 Mb sequence. This acts as input to the NIST statistical test suite. The binary stream generated by the hash function showed good p-values and pass rates for the relevant tests for hash functions as shown in table 1.

6.2 Performance Analysis

The proposed hash function has been implemented in C language, while the performance experiments were done on an Intel Core i5 (2.3 GHz) microprocessor. The speed of the hash function was calculated with a test file of size 10097 bytes and was found to be 1.1 MB/s. The speed of other hash functions computed in the same environment is given in Table 2.

The randomness property of the hash digest generated by the hash function is important to make it cryptographically secure. In order to get good p-values

Table 1. NIST Test Results

SI.No	Test Name	P-value	Status
1	Frequency test	0.419021	Pass
2	Block Frequency test	0.030806	Pass
3	Cumulative Sums test	0.935716	Pass
4	Runs test	0.236810	Pass
5	Longest Runs test	0.017912	Pass
6	Rank test	0.616305	Pass
7	FFT test	0.897763	Pass
8	Non overlapping template test	0.616305	Pass
9	Overlapping template test	0.514124	Pass
10	Random Excursions	0.911413	Pass
11	Random Excursions Variant	0.637119	Pass
12	Linear Complexity	0.719747	Pass

for the tests in NIST, we have tested the bit stream by increasing the number of cycles sequentially from 2 and found that 13 cycles gave good results. We have also implemented the same hash function without using omega flip permutation, i.e., the internal function of the sponge function has only the transformation step which made use of hybrid 3-neighbourhood CA rules. The speed achieved during that design was 6.84 MB/s which is much more than other popular hash functions. The design also showed good avalanche effect, but it did not pass some of the tests in NIST test suite with good p-values. While designing the new hash function, we have considered the randomness property and avalanche effect with equal priority and this accounts for the decreased speed of the hash. The size of the internal state bits of the sponge function, the values of r and c are also factors affecting the speed of the hash function. So, here we have a trade-off between security and speed. But, the reward for slow speed is the adjustable size of the hash digest due to the use of sponge functions and additional security due to CA and omflip permutation.

Table 2. Speed of Different Hash Functions

Hash function	Speed(MB/s)
SHA-224	4.52
SHA- 256	4.52
SHA-384	3.95
SHA-512	4.37
RIPEMD-160	4.13
WHIRLPOOL	4.49

7 Conclusion and Future Work

In this paper, we propose a new hash function based on Cellular Automata and sponge functions which takes an arbitrary length input and produces a 256 bit hash digest. The proposed function has tailored the best cryptographic primitives into the design and hence shows good Avalanche effect. We have used the CSHR

algorithm to develop a 3-neighbourhood CA rule set that uses both linear and non-linear rules. In addition, we have also used Omega Flip permutation which helps in permuting the bits in a fast and efficient manner. The proposed hash function produces a hash digest that resembles a random number, which makes it resistant to all known attacks on hash functions. We are still investigating the possibility of using other rule sets under CSHR algorithm. We are also analysing the impact of using CA rules of higher radii like 5-neighbourhood CA in place of 3-neighbourhood.

References

1. NIST Statistical Test Suite `https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software`, accessed: 2020-03-26
2. Aumasson, J.P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A lightweight hash. In: Mangard, S., Standaert, F.X. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2010. pp. 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
3. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: ECRYPT hash workshop. vol. 2007. Citeseer (2007)
4. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the indifferentiability of the sponge construction. In: Smart, N. (ed.) Advances in Cryptology – EUROCRYPT 2008. pp. 181–197. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. pp. 313–314. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
6. Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: Spongent: A lightweight hash function. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2011. pp. 312–325. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
7. Cannière, C., Dunkelman, O., Knežević, M.: Katan and ktantan – a family of small and efficient hardware-oriented block ciphers. In: Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems. p. 272–288. CHES '09, Springer-Verlag, Berlin, Heidelberg (2009)
8. Chakraborty, K., Chowdhury, D.R.: Cshr: Selection of cryptographically suitable hybrid cellular automata rule. In: Sirakoulis, G.C., Bandini, S. (eds.) Cellular Automata. pp. 591–600. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
9. Daemen, J., Govaerts, R., Vandewalle, J.: A framework for the design of one-way hash functions including cryptanalysis of damgård's one-way function based on a cellular automaton. In: International Conference on the Theory and Application of Cryptology. pp. 82–96. Springer (1991)
10. Damgård, I.B.: A design principle for hash functions. In: Conference on the Theory and Application of Cryptology. pp. 416–427. Springer (1989)
11. Gorski, M., Lucks, S., Peyrin, T.: Slide attacks on a class of hash functions. In: Pieprzyk, J. (ed.) Advances in Cryptology - ASIACRYPT 2008. pp. 143–160. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
12. Guo, J., Peyrin, T., Poschmann, A.: The photon family of lightweight hash functions. In: Rogaway, P. (ed.) Advances in Cryptology – CRYPTO 2011. pp. 222–239. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

13. Hammad, B.T., Jamil, N., Rusli, M.E., Reza, M.: A survey of lightweight cryptographic hash function. In: International Journal of Scientific and Engineering Research. vol. 8 (2017)
14. Hanin, C., Echandouri, B., Omary, F., Bernoussi, S.E.: L-cahash: A novel lightweight hash function based on cellular automata for rfid. In: UNet (2017)
15. Hell, M., Johansson, T., Maximov, A., Meier, W.: A stream cipher proposal: Grain-128. In: 2006 IEEE International Symposium on Information Theory. pp. 1614–1618 (2006)
16. Jamil, N., Mahmood, R.: A new cryptographic hash function based on cellular automata rules 30 134 and omega-flip network (2012)
17. Kuila, S., Saha, D., Pal, M., Chowdhury, D.R.: Cash: Cellular automata based parameterized hash. In: International Conference on Security, Privacy, and Applied Cryptography Engineering. pp. 59–75. Springer (2014)
18. Lee, R.B., Zhijie Shi, Xiiao Yang: Efficient permutation instructions for fast software cryptography. IEEE Micro **21**(6), 56–69 (Nov 2001)
19. Lee, R.B., Shi, Z.J., Yin, Y.L., Rivest, R.L., Robshaw, M.J.B.: On permutation operations in cipher design. In: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2 - Volume 2. p. 569. ITCC '04, IEEE Computer Society, USA (2004)
20. Matyas, S.M.: Generating strong one-way functions with cryptographic algorithm. IBM Technical Disclosure Bulletin **27**, 5658–5659 (1985)
21. Merkle, R.C.: One way hash functions and des. In: Conference on the Theory and Application of Cryptology. pp. 428–446. Springer (1989)
22. Mihaljevic, M., Zheng, Y., Imai, H.: A fast cryptographic hash function based on linear cellular automata over gf(q) (06 1998)
23. Mukhopadhyay, D., RoyChowdhury, D.: Cellular automata: An ideal candidate for a block cipher. In: Ghosh, R.K., Mohanty, H. (eds.) Distributed Computing and Internet Technology. pp. 452–457. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
24. Rivest, R.: The md5 message-digest algorithm (1992)
25. Sadak, A., Echandouri, B., Ziani, F.E., Hanin, C., Omary, F.: Lcahash-1.1: A new design of the lcahash system for iot. International Journal of Advanced Computer Science and Applications **10**(11) (2019)
26. Sadak, A., Ziani, F.E., Echandouri, B., Hanin, C., Omary, F.: Hcahf: A new family of ca-based hash functions. International Journal of Advanced Computer Science and Applications **10**(12) (2019)
27. Standard, S.H.: Fips publication 180-1. National Institute of Standards and Technology (1995)
28. Tomassini, M., Perrenoud, M.: Stream ciphers with one-and two-dimensional cellular automata. In: International Conference on Parallel Problem Solving from Nature. pp. 722–731. Springer (2000)
29. Wang, X., Yu, H.: How to break md5 and other hash functions. In: Cramer, R. (ed.) Advances in Cryptology – EUROCRYPT 2005. pp. 19–35. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
30. Wolfram, S.: Cryptography with cellular automata. In: Williams, H.C. (ed.) Advances in Cryptology — CRYPTO '85 Proceedings. pp. 429–432. Springer Berlin Heidelberg, Berlin, Heidelberg (1986)