Hash Function Design Based on Hybrid Five-Neighborhood Cellular Automata and Sponge Functions

Anita John
Jimmy Jose
Department of Computer Science and Engineering
National Institute of Technology Calicut
India

In today's world of pervasive computing, all the devices have become smart. The need for securing these devices becomes a need of the hour. The traditional cryptographic algorithms will not be ideal for small devices, and this opens a new area of cryptography named lightweight cryptography, which focuses on the implementation of cryptographic algorithms in resource-constrained devices without compromise in security. Cryptographic hash functions enable detection of message tampering by adversaries. This paper proposes a lightweight hash function that makes use of sponge functions and higher radii hybrid cellular automata (CAs). The proposed hash function shows good cryptographic properties as well as collision resistance and serves as an ideal hash function for lightweight applications.

Keywords: cryptographic hash functions; cellular automata; 5-neighborhood hybrid cellular automata; sponge functions; omegaflip permutation

1. Introduction

The last few decades saw a technological boom, and almost all devices are connected to the internet. While the devices became smart, the need for secure data storage and transmission gained more importance. Cryptographic hash functions are used to ensure data integrity and prevent data forgery. They are one-way functions that produce a message digest or hash digest from the input message. The hash digest serves as a unique fingerprint of the message since this short and secure code reveals any tampering made to that message. Most of the earlier cryptographic hash functions like MD5 and SHA made use of Merkle–Damgård [1, 2] and Davies–Meyer [3] schemes in their designs. But this era of pervasive computing demands lightweight algorithms that can be used in resource-constrained devices. The biggest challenge in the design of lightweight hash functions is the tradeoff

https://doi.org/10.25088/ComplexSystems.32.2.171

between security and memory requirements. The sponge functions and cellular automata (CAs) have marked their places in lightweight applications. We proposed a hash function using three-neighborhood (3-N) hybrid CAs and sponge functions in [4]. In this paper, we replace the 3-N cellular automaton (CA) with a higher-radii hybrid CA.

The paper is organized as follows: Section 2 discusses CAs, five-neighborhood (5-N) CAs and sponge functions. Section 3 provides the state of the art in sponge function– and CA-based hash functions, followed by the working of the proposed hash function. Sections 4 and 5 discuss the design rationale and security analysis of the hash function, followed by conclusions in Section 6.

2. Preliminaries

This section gives a brief description of CAs and sponge functions, which serve as the building blocks of our hash function.

2.1 Introduction to Cellular Automata

A CA can be viewed as a lattice of cells that stores binary values. The CA will be initialized with 0s and 1s. The values in the cells get updated at every clock cycle based on a transition function called the CA rule. During each cycle, the value of a cell gets updated based on the values of its neighboring cells, which serve as the input to the CA rule. So a dependency exists among the neighboring cells in CAs. Every cell gets updated in parallel during each clock cycle. CA rules can be classified as linear and nonlinear. The linear rules use only the XOR operation in their combinational logic, and nonlinear rules use AND/OR operations in addition to the XOR operation in their logic. In general, the number of neighboring cells n that are involved in a CA rule is given by $n = 2r + 1$, where r is the radius of the neighborhood. CAs can further be classified as uniform and hybrid based on whether all the cells use the same CA rule or different CA rules.

The boundary condition is another factor that decides the properties of the CA. The boundary conditions decide the leftmost and rightmost neighbor cells of the CA. This paper uses the *one null boundary* condition, which takes 1 as the neighbor of the leftmost cell and 0 as the neighbor of the rightmost cell. This boundary condition was found to make the hybrid rule set cryptographically robust [5]. This motivated us to use one null boundary condition. The parallel execution of CAs makes them ideal for lightweight applications. The neighborhood radii of the CA have an impact on the strength of the

CA when used in secure applications. Most of the work based on CAs used 3-N CAs, and recently some applications have used 5-N CA rules.

2.1.1 Five-Neighborhood Cellular Automaton Rules

Most of the work related to CAs has used the elementary (3-N) CA. The rate of diffusion of CA-based pseudorandom number generators (PRNGs) increases with the increase in neighborhood radii [6]. The cryptographic properties of bipermutive rules of radius two were explored in [6] and a study of 5-N linear hybrid CAs is in [7]. A set of cryptographically suitable 5-N CA rules was selected based on these studies. The rule numbers are taken from [6]. Figure 1 shows the neighbors of a cell s_i in 5-N CA.

S_{i-2}	S_{i-1}	S_i	S_{i+1}	S_{i+2}

Figure 1. Neighboring cells in five-neighborhood CA.

The algebraic normal form (ANF) of the 5-N CA rules used in our hash function is

$$\text{Rule 1721342310: } s_{i-2} \oplus s_{i-1} \oplus s_{i+1} \oplus s_{i+2}$$

$$\text{Rule 2523490710: } s_{i-2} \oplus s_{i-1} \oplus s_i \oplus s_{i+1} \oplus s_{i+2}$$

$$\text{Rule 1452976485: } s_{i-2} \oplus s_{i-1} \cdot s_i \oplus s_{i-1} \cdot s_{i+1} \oplus s_i \oplus s_{i+2} \oplus 1$$

$$\text{Rule 1520018790: } s_{i-2} \oplus s_{i-1} \cdot s_{i+1} \oplus s_i \cdot s_{i+1} \oplus s_{i+1} \oplus s_{i+2}$$

where s_{i-2} and s_{i-1} represent the second-left and left neighbor of s_i, respectively. s_{i+2} and s_{i+1} represent the second-right and right neighbor of s_i, respectively. $\oplus$ represents the XOR operation and $\cdot$ represents the AND operation.

Rules 1721342310 and 2523490710 are 5-N linear CA rules and rules 1452976485 and 1520018790 are 5-N nonlinear CA rules. Here, we have followed a hybrid rule set consisting of linear and nonlinear 5-N CA rules. This paper follows the order of rules in the rule set from the CSHR algorithm that was proposed for elementary CA rules in [5]. In [8], Sandip et al. proposed a cryptographically robust hybrid rule set. Any cryptographic application needs to have properties like algebraic degree, nonlinearity and correlation immunity. The nonlinear CA rules help achieve algebraic degree and nonlinearity, but they lack correlation immunity. Linear CA rules, on the other hand, are correlation immune, but lack algebraic degree and nonlinearity properties. So the linear and nonlinear CA rules, when used

together, are likely to give good cryptographic properties, and an analytical argument is provided in [8] to show that the introduction of linear rules helps reduce correlation after some iterations. The paper considered five rule sets, where every rule set contained the nonlinear CA rule 30, which showed good pseudorandom properties in addition to balance. These rule sets incorporated rule 30 with several other linear and nonlinear elementary CA rules and analyzed the cryptographic suitability of all these rule sets using a new test called the *d-monomial test*.

In [5] Kaushik et al. proposed an algorithm for generating cryptographically suited hybrid rules (CSHR) using elementary CA rules. The same algorithm was extended to 5-N CAs to construct a rule set using cryptographically suitable 5-N CA rules. The rule set thus formed is

$$\langle 1452976485, 1721342310, 2523490710, 1452976485, 1520018790, 1452976485, 1721342310, 2523490710\rangle.$$

These rules are expected to exhibit good algebraic degree, nonlinearity and period based on the intuition that the cryptographic properties will become better with the increase in neighborhood radii. This was seen after their use in the CA-based stream ciphers CARPenter [9] and Pentavium [10].

2.2 Sponge Functions in the Design of Cryptographic Hash Functions

Cryptographic hash functions are one-way functions that serve as a short compressed string derived from the message that depends on all the bits of the message in an unpredictable manner. They take an arbitrary-length input and produce a fixed-length output. Any cryptographic hash function must satisfy the following properties: collision resistance, preimage resistance and second preimage resistance. The hash functions MD5 [11] and SHA [12] used during the early 1980s and late 1990s made use of compression functions iteratively. They were later cryptanalyzed and found to be insecure [13]. NIST called for an open competition for a new and efficient family of hash functions named SHA-3, and in 2012 Keccak [14] was declared the winner.

While SHA was built on the Merkle–Damgård construction, Keccak was based on a new design approach named sponge functions developed by Bertoni et al. [15]. In sponge functions, the compression function is replaced by an internal transformation function. The input message is first absorbed into the sponge through a series of transformations and then squeezed out to release the hash digest, hence the name "sponge" functions. These operations have two phases: the

absorbing phase and the squeezing phase, and the same transformation function is used in both phases. The input to the transformation function is split into r and c, where r forms the bit rate and c forms the capacity. The input message is partitioned into blocks of r bits. The smaller the value of r, the larger the number of message blocks getting diffused into the final hash digest will be, and the larger the number of internal transformations will be. This improves the strength of the hash function at the expense of more execution time. The security level that can be attained using sponge functions is dependent on c [16].

2.2.1 Working of Sponge Function

The input message is preprocessed and divided into blocks of r bits. Each of the message blocks passes through the following phases:

Absorbing phase. This phase absorbs all the message blocks into the sponge. We define an initial state of b zeros where $b = r + c$. Each r-bit message block is XORed with r bits of the b state bits and is subjected to the internal transformation function, which executes for n_r rounds. The output of the first transformation function serves as the state bits for the next iteration of the transformation function. The second r-bit message block will be XORed with r bits of this state and executes for the next n_r rounds. This continues until all the message blocks get XORed. So if an input message is divided into d blocks of r bits during the preprocessing phase, the transformation function will be invoked d times, and during each invocation, it executes for n_r rounds. The number of transformation steps in the absorbing phase depends on the value of r; the smaller the value of r, the larger the capacity c of the sponge, and in turn the larger the number of iterations of the transformation function.

The absorbing phase ensures that all the bits of the input message are dispersed well among themselves, which ensures the diffusion of bits. At the end of this phase, all the message blocks will be "absorbed" into the sponge. This marks the end of the absorbing phase.

Squeezing phase. This phase "squeezes out" the hash digest. This phase makes use of the same transformation function F used in the absorbing phase except the XORing of the message block. The output of the absorbing phase undergoes transformation of n_r rounds, and r bits will be squeezed out. This continues until the required length of the hash digest has been squeezed out. The length of the hash digest will be a multiple of r. The sponge function provides the flexibility of truncating the same to get the desired length, which is a feature unique to sponge functions and was not provided by the earlier schemes.

https://doi.org/10.25088/ComplexSystems.32.2.171

3. Proposed Hash Function Based on Hybrid Five-Neighborhood Cellular Automata and Sponge Functions

3.1 Hash Functions Based on Cellular Automata and Sponge Functions: An Overview

The use of CAs in cryptographic hash functions had been proposed by Damgård in [2]. This was cryptanalyzed by Daemen [17] and he proposed Cellhash, which made use of CAs with periodic boundary conditions. Several other CA-based hash functions were proposed by Mihaljevic et al. [18], Hanin et al. [19, 20], Jamil et al. [21] and Sadak et al. [22]. All the hash functions followed the Merkle–Damgård construction using hybrid CAs inside the compression function.

Sponge functions have been used in lightweight applications. Some of the sponge function–based hash functions include Quark [23], SPONGENT [24] and PHOTON [25]. A detailed description of all the lightweight cryptographic hash functions is found in B. Hammad et al. [26]. The strength of sponge functions can be enhanced by strengthening the internal transformation function. This motivated researchers to incorporate the lightweight primitive CA into the function. Keccak [14], the SHA-3 winner, uses a CA-incorporated sponge function that affects the two neighborhood cells of each cell in its χ transformation. Another CA-infused sponge function was used in the CASH [27] family of hash functions. CASH made use of tweakable parameters, which makes it useful for different applications. Most of the CA-based sponge function constructions use 3-N CA in uniform and linear hybrid versions. A 3-N hybrid rule set together with the omegaflip permutation is used in [4]. The use of a higher-radii CA inside the sponge function is the first of its kind.

In this paper, we have replaced the 3-N hybrid rule set used in [4] with a 5-N hybrid rule set, along with a permutation layer that makes use of the omegaflip permutation [28]. The use of 5-N CA rules together with permutation should increase the strength of the internal function and hence add to the strength of the hash function against attacks. The values of r and c can be decided based on the level of security and memory requirements of the application. In this paper, we use $r = 92$ and $c = 132$. Figure 2 shows the structure of the hash function.

3.2 Working of the Proposed Hash Function

The proposed hash function proceeds through the following steps:

Preprocessing phase. This phase includes padding and partitioning of the input message. The padding step makes the length of the arbitrarily long message a multiple of r. The 64-bit representation of the input message length preceded by "10" serves as the padding bits.

This also helps enhance the collision resistance property. The padded message is then partitioned into r-bit blocks $M_1, M_2, \ldots, M_i$.

Absorbing and squeezing phase. The absorbing phase proceeds by applying the internal transformation function on each message block M_i. The state bits of length b are initially set to all zeros. The first message block M_1 is XORed with the r bits of the b state bits and these XORed r bits together with c bits will be transformed by the internal transformation function. The second message block M_2 is XORed with the r bits of the output of the first n_r rounds of the transformation function. This continues for all remaining blocks. After the processing of all message blocks, an additional round of transformation without any XOR input is done to prevent sliding attacks on the hash function. The block diagram of the internal function F of the sponge function is shown in Figure 3.

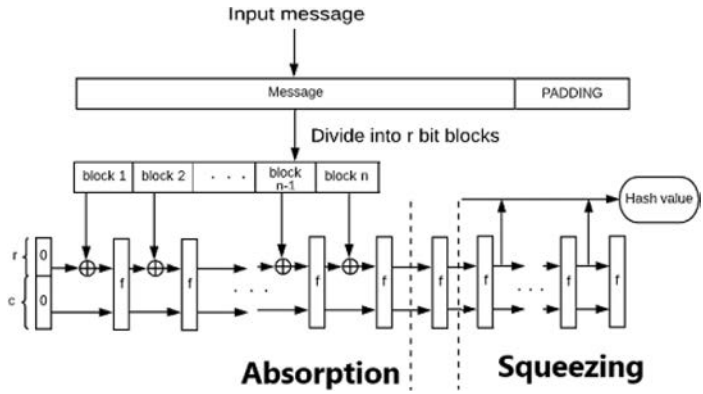

Figure 2. Structure of hash function.

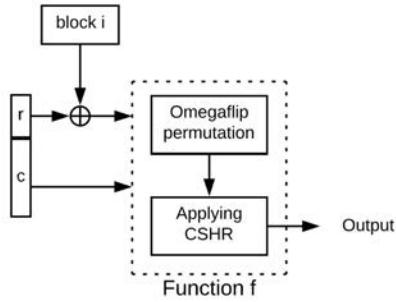

Figure 3. Internal function.

https://doi.org/10.25088/ComplexSystems.32.2.171

3.2.1 Working of the Internal Transformation Function

There are two steps in the transformation function: permutation and CA-based transformation. We have used the omegaflip permutation to permute the state bits. The order of performing the 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. This is followed by the CA-based transformation, where the 5-N hybrid CA rules are applied on the permuted state bits. The CA evolves through eight cycles during each round of the internal transformation function and there are two rounds (i.e., $n_r = 2$) for the internal function.

The output from the absorbing phase undergoes two rounds of the internal transformation function during the squeezing phase. This function resembles the function in the absorbing phase except the XOR operation with the message block. The r bits from this function are "squeezed out" to form part of the hash digest. During this phase, the b bits of the state are taken as the control bits for the omegaflip permutation. The number of iterations of the internal transformation function depends on the expected size of the hash digest. The length of the hash digest is a multiple of the bit-rate r. If the length exceeds the desired length, it can be truncated to the required length. The steps of the hash function are described in Algorithm 1.

Input: Message M
Output: 256-bit message digest

begin
Divide the message M of length L into blocks M_1, M_2, $\ldots M_n$ of r bits.

If L is not a multiple of r, pad the message with "10" followed by the 64-bit representation of the message length until L becomes a multiple of r.

Initialize a block $b = r \,||\, c$ with zeros, $r_0 = 0$ and $c_0 = 0$.

//Absorbing phase
for $i \rightarrow 1$ to n **do**
// Single round of internal transformation
$r_i = M_i \oplus r_{i-1}$
$c_i = c_{i-1}$
$b_i =$ omega flip (b_i) //Permute the entire b block using omegaflip permutation
$b_i = \mathrm{CA}_5(b_i)$ // Apply the 5-N hybrid CA rules to each of the cells in that order
end for
Add a Null round, that is, one round of internal transformation without XORing with any message block.
$b_n = r_n \,||\, c_n$ will be the output after the absorbing phase and this will be given as input to the squeezing phase.

// Squeezing phase
Repeat the steps until 256-bit hash digest is squeezed out.

for $i \rightarrow (n+1)$ to $(n+3)$ **do**
$b_i =$ omega flip (b_i)
$b_i = \mathrm{CA}_5\ (b_i)$
Append r_i to the final hash digest.
end for

Truncate the final output to get the final 256-bit message digest.
end

Algorithm 1. Hash function using 5-N CAs and sponge function.

4. Design Rationale for Using Cellular Automata in Sponge Functions

While most of the earlier cryptographic hash functions followed the Merkle–Damgård (MD) scheme, the last decade ushered in the need for lightweight functions in place of the traditional compression function. This helped to thwart the length extension attacks and multicollision attacks prevalent on MD schemes. The idea of integrating sponge functions and CAs into the design of our new hash function evolved as a result of this. The next aim was to add to the strength of the internal function without compromising the lightweight property of a sponge function. This resulted in the combination of higher-radii hybrid CAs and sponge functions.

The motivation for using the sponge function in our design is:

1. resemblance to random oracle
2. flexibility in the length of the hash digest
3. low memory requirements
4. absence of feed-forward mechanism

4.1 Suitability of Five-Neighborhood Cellular Automata inside a Sponge Function

Some properties of CAs that make them an ideal primitive for secure and lightweight hash functions follow.

4.1.1 Chaotic Nature

The CA rules were classified into four classes based on the nature of patterns generated by the rules during time evolution. This shows the statistical behavior of the rules [29]. Rules 30, 90 and 150 belong to

class 3 and generate chaotic patterns, hence are ideal for pseudorandom number generation. The 5-N CA rules selected in the rule set used in our hash function are bipermutive rules. These rules are as good as the elementary rule 30 and in addition, they satisfy the property of chaos [6]. This chaotic nature of the CA rules makes them ideal in the design of hash functions.

4.1.2 Hardware Efficiency and Parallelism

All the CA operations execute in parallel and involve only simple logical operations. This makes CAs suitable in software and hardware and an ideal lightweight primitive in resource-constrained devices and applications. Hence CA-based hash functions are likely to gain more focus in internet of things (IoT) devices, blockchain and elsewhere.

The diffusion rate and other cryptographic properties of the internal transformation function can be enhanced by the use of higher-radii CA rules. The properties of sponge function–based hash functions solely depend on the internal transformation function. The hash digest produced from a message has to be dependent on all the bits of the message in order to improve its diffusion rate and collision resistance. This motivated us to introduce 5-N linear and nonlinear rules in place of elementary CA rules. The difference between 3-N and 5-N CA rules is the number of bits that get involved in the state update function of a single cell; while 3-N rules depend on $2q + 1$ cells in a single iteration, 5-N CA rules depend on $4q + 1$ cells, where q denotes the number of cycles.

We have used a hybrid rule set that consists of linear and nonlinear 5-N CA rules. The use of 5-N CA rules improves the diffusion rate of the CA, since during each cycle, the number of bits involved in the state update function of the CA is more when compared to 3-N CAs. Cryptographic hash functions are expected to have good cryptographic properties and collision resistance and produce a hash digest that resembles a random number. The application of higher-radii hybrid CA rules and permutation in a single internal round of the sponge function ensures the dependency of all the bits in a single block of the message. The absorbing phase ensures that the bits among different message blocks are also mixed well. This high dependency of all the message bits with the output is the major strength of the hash function against probable collision attacks.

5. Security and Performance Analysis

The security criteria to be satisfied by cryptographic hash functions that produce an n-bit hash digest are:

1. *Determinism*. Each input message should generate the same hash digest whenever it is subjected to the same hash function.
2. *Collision resistance*. It should be computationally infeasible (in less than $2^{n/2}$ work) to find any two random messages that produce the same hash digest.
3. *Preimage resistance*. Given only the hash value, it should not be possible to find a message that generates that hash in less than about 2^n work.
4. *Second preimage resistance*. Given a message m and a hash function H, it should not be possible to find another message m' that yields the same hash digest in less than about 2^n work.

These properties were theoretically evaluated to assess the strength of the proposed hash function against cryptanalytic attacks.

5.1 Confusion and Diffusion Effect

Shannon's confusion and diffusion were defined for encryption algorithms [30]. The definitions of confusion and diffusion need to be modified in the case of cryptographic hash functions to make them suitable for the functions' design. Confusion ensures that the relationship between the input message and the hash digest is unpredictable. The diffusion property of the hash function ensures that each of the input bits is dispersed into the final hash digest such that a change in a single input bit gets reflected in the hash digest in terms of the number of altered bits.

In order to assess these properties of our proposed hash function, we considered a random message and computed its hash digest. A variant of the original message was made by changing a random bit, and its hash digest was also computed. The hash digests thus calculated were compared for the number of bits that were flipped. This was done for 100 messages, and in all the cases, more than half of the bits of the hash digest were altered, even with a single bit change. This property is called the avalanche effect.

Figure 4 illustrates the avalanche effect of the proposed hash function. The x axis represents the number of bits changed in the input and the y axis represents the Hamming weight. The graph shows the number of bits altered in the hash digest when small changes are made in the input messages. In all cases, more than half the bits of the hash digest were altered.

A sample of the hash digest computed for an input message and its variant is as follows:

Input message: This is a new hash function

Hash digest: F0D060228EF61CAF952517E5FC21DC-
CD1E7622E423ABD319E041012032F95A43

We now add a "." at the end of the previous sentence.

Input message: This is a new hash function.

Hash Digest: B91136FDED5E68C1F030A2E6E2E0521CA8C55F-B876506721A307D2B7303262A2

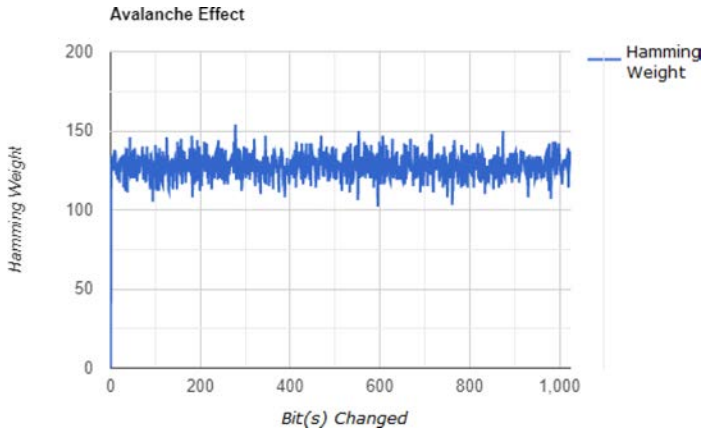

Figure 4. Avalanche effect.

5.2 Theoretical Security Analysis

The use of sponge functions in the design of hash functions should be enough to claim collision resistance, preimage resistance and second preimage resistance, since sponge functions behave like a random oracle. The use of a CA and permutation in the internal transformation function of a sponge function make it stronger. The security of a hash function relies on the difficulty for an adversary to find another message with the same hash digest with or without the knowledge of the input message. This attempt will be thwarted firsthand, since the hash digest is of length 256 bits. As seen in the previous example, a single change in the input is well reflected in the output, which adds to the resistance of collision. Moreover, the use of 5-N CAs adds to the diffusion of message bits into the hash digest, since the number of bits involved during each cycle is more when compared to 3-N CAs.

Given a hash digest output of size n, we need to perform 2^n operations to find a preimage or a second preimage and $2^{n/2}$ operations to find a collision [2]. During the CA-based transformation in the internal function, each bit becomes dependent on 65 neighboring bits at the end of 16 CA cycles (8 CA cycles × 2 rounds), since we are using 5-N CAs and each bit is dependent on $4q + 1$ neighboring bits where q is the number of cycles of the CA. The hash digest is produced after many rounds of internal transformation involving CA transformation and permutation during the absorbing and squeezing phases. This

ensures the diffusion of input into the resultant hash digest and hence resistance to preimage, second preimage and collision attacks.

5.3 Randomness Test Using NIST Statistical Test Suite

Ideally, a hash digest should be a (pseudo) random number, and this randomness property is necessary for its cryptographic applications. In order to assess the randomness quality of the generated hash digest, we have used the NIST Statistical Test Suite [31]. The NIST test suite consists of 15 empirical tests based on statistical hypothesis testing. The results of these tests are shown in the form of P-values. Each P-value is the probability that a perfect random number generator would have produced a sequence with the same randomness quality as the sequence that was tested, or with less randomness quality than the sequence. A P-value of 0 indicates nonrandomness and a P-value of 1 indicates perfect randomness [32].

The input to the NIST test suite is generated by applying the hash function H on an initial seed S that is incremented by 1. That is, we calculate $H(S)$, $H(S+1)$ and so on until we are able to concatenate them to get a 100 Mb sequence. The binary stream generated by the hash function showed good P-values and pass rates for the relevant tests for hash functions and is shown in Table 1. Assessing the randomness quality of the hash digest affirms the cryptographic strength of the hash function against known attacks.

SI. No	Test Name	P-value	Status
1	frequency test	0.319084	pass
2	block frequency test	0.162606	pass
3	cumulative sums test	0.145326	pass
4	runs test	0.657933	pass
5	longest runs test	0.096578	pass
6	rank test	0.350485	pass
7	FFT test	0.779188	pass
8	non-overlapping template test	0.779188	pass
9	overlapping template test	0.048716	pass
10	serial	0.058984	pass
11	linear complexity	0.289667	pass

Table 1. NIST test results.

The proposed hash function has been implemented in the C programming language, while the performance experiments were done on an Intel Core i5 (2.3 GHz) microprocessor. The speed of the hash

function is shown in Table 2. The size of the internal state bits of the sponge function and the values of r and c are factors that affect the speed of the hash function. So here we have a tradeoff among security, speed and resource requirements. In practical applications, 3-N CA-based hash functions will be preferable if the hardware is limited. The use of 5-N CAs helps to reduce the execution time, since the diffusion of bits can be achieved in a smaller number of cycles than 3-N CAs. A more optimized implementation will definitely help reduce the memory requirements and also increase the speed.

Parameter	3-N	5-N
execution time (in sec)	171.74	118.492
processing speed (Mb/sec)	0.0569	0.0821
average Hamming distance	134	130
number of cycles	13	8

Table 2. Comparison of speed of 5-N and 3-N hybrid CAs.

6. Conclusion

In this paper, a cryptographic hash function based on sponge functions and higher-radii hybrid cellular automata (CAs) has been proposed. The design has brought together the best cryptographic primitives for lightweight applications. We have used cryptographically suitable linear and nonlinear five-neighborhood (5-N) cellular automaton (CA) rules and omegaflip permutation in the internal function of the sponge function construction. We have analyzed the collision resistance properties of the hash function and found it to perform well. In addition, we checked the randomness properties of the hash digest using an NIST test suite and it showed statistical similarity with a random number, which adds to the strength of the function. There is a tradeoff between security and memory requirements, and the use of higher-radii CA rules can be decided based on that. As a future direction, we plan to implement this in hardware and also to replace the permutation function used in the internal transformation with a CA-based S-box so as to strengthen the internal function.

References

[1] R. C Merkle, "One Way Hash Functions and DES," in *Advances in Cryptology (CRYPTO '89)* (G. Brassard, ed.), New York: Springer, 1989 pp. 428–446. doi:10.1007/0-387-34805-0_40.

[2] I. B. Damgård, "A Design Principle for Hash Functions," in *Advances in Cryptology (CRYPTO '89)* (G. Brassard, ed.), New York: Springer, 1989 pp. 416–427. doi:10.1007/0-387-34805-0_39.

[3] B. Preneel, R. Govaerts and J. Vandewalle, "Hash Functions Based on Block Ciphers: A Synthetic Approach," in *Advances in Cryptology: 13th Annual International Cryptology Conference (CRYPTO'93)*, Santa Barbara, CA (D. R. Stinson, ed.), Berlin, Heidelberg: Springer, 1993 pp. 368–378. doi:10.1007/3-540-48329-2_31.

[4] A. John, A. Reji, A. P. Manoj, A. Premachandran, B. Zachariah and J. Jose, "A Novel Hash Function Based on Hybrid Cellular Automata and Sponge Functions," in *Proceedings of First Asian Symposium on Cellular Automata Technology*, Kolkata, India (S. Das and G. J. Martinez, eds.), Singapore: Springer Nature, 2022 pp. 221–233. doi:10.1007/978-981-19-0542-1_16.

[5] K. Chakraborty and D. R. Chowdhury, "CSHR: Selection of Cryptographically Suitable Hybrid Cellular Automata Rule," in *Cellular Automata (ACRI 2012)* Santorini Island, Greece (G. Ch. Sirakoulis and S. Bandini, eds.), Berlin, Heidelberg: Springer, 2012 pp. 591–600. doi:10.1007/978-3-642-33350-7_61.

[6] A. Leporati and L. Mariot, "Cryptographic Properties of Bipermutive Cellular Automata Rules," *Journal of Cellular Automata*, **9**(5–6), 2014 pp. 437–475.

[7] S. Maiti and D. Roy Chowdhury, "Study of Five-Neighborhood Linear Hybrid Cellular Automata and Their Synthesis," in *Mathematics and Computing, Third International Conference (ICMC 2017)*, Haldia, India (D. Giri, R. N. Mohapatra, H. Begehr and M. S. Obaidat, eds.), Singapore: Springer, 2017 pp. 68–83. doi:10.1007/978-981-10-4642-1_7.

[8] S. Karmakar, D. Mukhopadhyay and D. Roy Chowdhury, "*d*-monomial Tests of Nonlinear Cellular Automata for Cryptographic Design," in *Cellular Automata: 9th International Conference on Cellular Automata for Research and Industry (ACRI 2010)*, Ascoli Piceno, Italy (S. Bandini, S. Manzoni, H. Umeo and G. Vizzari, eds.), Berlin, Heidelberg: Springer, 2010 pp. 261–270. doi:10.1007/978-3-642-15979-4_28.

[9] R. Lakra, A. John and J. Jose, "CARPenter: A Cellular Automata Based Resilient Pentavalent Stream Cipher," in *Cellular Automata: 13th International Conference on Cellular Automata for Research and Industry (ACRI 2018)*, Como, Italy (G. Mauri, S. El Yacoubi, A. Dennunzio, K. Nishinari and L. Manzoni, eds.), Cham: Springer, 2018 pp. 352–363. doi:10.1007/978-3-319-99813-8_32.

[10] A. John, B. C. Nandu, A. Ajesh and J. Jose, "PENTAVIUM: Potent Trivium-like Stream Cipher Using Higher Radii Cellular Automata," in *Cellular Automata: 14th International Conference on Cellular Automata for Research and Industry (ACRI 2020)*, Lodz, Poland

(T. M. Gwizdałła, L. Manzoni, G. Ch. Sirakoulis, S. Bandini and K. Podlaski, eds.), Cham: Springer, 2021 pp. 90–100. doi:10.1007/978-3-030-69480-7_10.

[11] R. Rivest. "The MD5 Message Digest Algorithm." 1992. datatracker.ietf.org/doc/html/rfc1321.

[12] "Secure Hash Standard, FIPS Publication 180-1," Gaithersburg, MD: National Institute of Standards and Technology, 1995. csrc.nist.gov/publications/detail/fips/180/1/archive/1995-04-17.

[13] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in *Advances in Cryptology: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2005)*, Aarhus, Denmark (R. Cramer, ed.), Berlin, Heidelberg: Springer, 2005 pp. 19–35. doi:10.1007/11426639_2.

[14] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "Keccak," in *Advances in Cryptology: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2013)*, Athens, Greece (T. Johansson and P. Q. Nguyen, eds.), Berlin, Heidelberg: Springer, 2013 pp. 313–314. doi:10.1007/978-3-642-38348-9_19.

[15] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "Sponge Functions," in *ECRYPT Hash Workshop 2007*, Semantics Scholar, 2007. pdfs.semanticscholar.org/0338/0dd678b5dbf37734452ac57f793db1a9620c.pdf.

[16] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "On the Indifferentiability of the Sponge Construction," in *Advances in Cryptology: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2008)*, Istanbul, Turkey (N. Smart, ed.), Berlin, Heidelberg: Springer, 2008 pp. 181–197. doi:10.1007/978-3-540-78967-3_11.

[17] J. Daemen, R. Govaerts and J. Vandewalle, "A Framework for the Design of One-Way Hash Functions Including Cryptanalysis of Damgård's One-Way Function Based on a Cellular Automaton," in *Advances in Cryptology: International Conference on the Theory and Application of Cryptology (ASIACRYPT '91)*, Fujiyoshida, Japan (H. Imai, R. L. Rivest and T. Matsumoto, eds.), Springer, 1991 pp. 82–96. doi:10.1007/3-540-57332-1_7.

[18] Y. Zheng, H. Imai and M. Mihaljevic, "A Fast Cryptographic Hash Function Based on Linear Cellular Automata over GF(q)," in *Global IT Security: Proceedings of the International Federation for Information Processing TC11 14th International Conference on Information Security (SEC '98)*, Austrian Computer Society, 1998 pp. 96–107. www.researchgate.net/publication/2790711_A_Fast_Cryptographic_Hash_Function_Based_on_Linear_Cellular_Automata _over _GFq.

[19] C. Hanin, B. Echandouri, F. Omary and S. El Bernoussi, "L-CAHASH: A Novel Lightweight Hash Function Based on Cellular Automata for RFID," in *Ubiquitous Networking, Third International Symposium (UNet 2017)*, Casablanca, Morocco (E. Sabir, A. G. Armada, M. Ghogho and M. Debbah, eds.) Cham: Springer, 2017 pp. 187–298. doi:10.1007/978-3-319-68179-5_25.

[20] A. Sadak, B. Echandouri, F. E. Ziani, C. Hanin and F. Omary, "LCAHASH-1.1: A New Design of the lCAHASH System for IoT," *International Journal of Advanced Computer Science and Applications*, **10**(11), 2019. doi:10.14569/IJACSA.2019.0101134.

[21] N. Jamil, R. Mahmood, M. Z'aba N. Udzir and Z. A. Zukarnaen, "A New Cryptographic Hash Function Based on Cellular Automata Rules 30, 134 and Omega-Flip Network." 2012. www.semanticscholar.org/paper/A-New-Cryptographic-Hash-Function-Based-on-Cellular-Jamil-Mahmood/1563c58daf7a9f63784ee19a98aeb529b7eb66bb.

[22] A. Sadak, F. E. Ziani, B. Echandouri, C. Hanin and F. Omary, "HCAHF: A New Family of CA-based Hash Functions," *International Journal of Advanced Computer Science and Applications*, **10**(12), 2019. doi:10.14569/IJACSA.2019.0101267.

[23] J.-P. Aumasson, L. Henzen, W. Meier and M. Naya-Plasencia, "QUARK: A Lightweight Hash," in *Cryptographic Hardware and Embedded Systems: 12th International Workshop (CHES 2010)*, Santa Barbara, USA (S. Mangard and F.-X. Standaert, eds.), Berlin, Heidelberg: Springer, 2010 pp. 1–15. doi:10.1007/978-3-642-15031-9_1.

[24] A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varıcı and I. Verbauwhede, "SPONGENT: A Lightweight Hash Function," in *Cryptographic Hardware and Embedded Systems: 13th International Workshop (CHES 2011)*, Nara, Japan (B. Preneel and T. Takagi, eds.), Berlin, Heidelberg: Springer, 2011 pp. 312–325. doi:10.1007/978-3-642-23951-9_21.

[25] J. Guo, T. Peyrin and A. Poschmann, "The PHOTON Family of Lightweight Hash Functions," in *Advances in Cryptology: 31st Annual Cryptology Conference (CRYPTO 2011)*, Santa Barbara, CA (P. Rogaway, ed.), Berlin, Heidelberg: Springer, 2011 pp. 222–239. doi:10.1007/978-3-642-22792-9_13.

[26] B. T. Hammad, N. Jamil, M. E. Rusli and M. Reza, "A Survey of Lightweight Cryptographic Hash Function," in *International Journal of Scientific and Engineering Research*, **8**(7), 2017. www.ijser.org/researchpaper/A-survey-of-Lightweight-Cryptographic-Hash-Function.pdf.

[27] S. Kuila, D. Saha, M. Pal and D. Roy Chowdhury, "CASH: Cellular Automata Based Parameterized Hash," in *Security, Privacy, and Applied Cryptography Engineering: 4th International Conference (SPACE 2014)*, Pune, India (R. S. Chakraborty, V. Matyas and P. Schaumont, eds.), Cham: Springer, 2014 pp. 59–75. doi:10.1007/978-3-319-12060-7_5.

[28] R. B. Lee, Z. Shi and X. Yang, "Efficient Permutation Instructions for Fast Software Cryptography," *IEEE Micro*, **21**(6), 2001 pp. 56–69. doi:10.1109/40.977759.

[29] G. J. Martínez, J. C. Seck-Tuoh-Mora and H. Zenil, "Wolfram's Classification and Computation in Cellular Automata Classes III and IV," *Irreducibility and Computational Equivalence: 10 Years After Wolfram's A New Kind of Science* (H. Zenil, ed.), Berlin, Heidelberg: Springer, 2013 pp. 237–259. doi:10.1007/978-3-642-35482-3_17.

[30] C. E. Shannon, "Communication Theory of Secrecy Systems," *The Bell System Technical Journal*, **28**(4), 1949 pp. 656–715. doi:10.1002/j.1538-7305.1949.tb00928.x.

[31] "NIST SP 800-22: Download Documentation and Software." Gaithersburg, MD: National Institute of Standards and Technology. csrc.nist.gov/projects/random-bit-generation/documentation-and-software.

[32] "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," Technical Report NIST Special Publication (SP) 800-22, Gaithersburg, MD: National Institute of Standards and Technology, 2010. doi:10.6028/NIST.SP.800-22.