IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 20, 2020, accepted December 7, 2020, date of publication January 8, 2021, date of current version February 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3049881

A Novel Hash Function Based on a Chaotic Sponge and DNA Sequence

MOATSUM ALAWIDA[1], AZMAN SAMSUDIN[1], NANCY ALAJARMEH[2], JE SEN TEH[1], MUSHEER AHMAD[3], AND WAFA' HAMDAN ALSHOURA[1]

[1]School of Computer Sciences, Universiti Sains Malaysia, Gelugor 11800, Malaysia
[2]Department of Computer Science, Faculty of Information and Communications Technology, Tafila Technical University, Tafila 66110, Jordan
[3]Department of Computer Engineering, Jamia Millia Islamia, New Delhi 110025, India

Corresponding authors: Moatsum Alawida (matsm88@yahoo.com) and Azman Samsudin (azman.samsudin@usm.my)

This work was supported in part by Universiti Sains Malaysia under Grant LRGS/MRUN:203.PKOMP.6777005.

ABSTRACT Many chaos-based hash functions have convoluted designs that are not based on proper design principles, complicating the verification of security claims. We address this problem by proposing a hash function based on a chaotic sponge construction and DNA sequence. DNA sequence is used to design state transition rules of a deterministic chaotic finite-state automata (DCFSA), a chaotic structure that enhances the chaoticity of digital chaotic maps. We use a DCFSA configuration consisting of four states associated with logistic maps. Analysis shows that the DCFSA configuration is efficient and has high chaotic complexity. Both DCFSA and DNA are used to design the sponge-based hash function. An input message is transformed into a DNA sequence and divided into fixed-length blocks. Each block is absorbed into a sponge structure via DNA-XOR. The sponge state is then updated with a new DNA sequence by iterating DCFSA as a compression function. Statistical evaluations indicate that the proposed hash function has near-ideal diffusion, confusion, collision resistance and distribution properties. The proposed hash can also extend the provably secure notions of the sponge construction, specifically the indifferentiability from random oracles. Furthermore, the hash function has a high level of performance as compared to other chaos-based hash functions.

INDEX TERMS Chaotic map DNA data integrity finite state automata hash function security.

I. INTRODUCTION

Chaos theory is the study of dynamical systems which appear to be random or irregular, but is strictly deterministic. On the other hand, digital chaos is the simulation or execution of these deterministic nonlinear dynamical systems on a fixed-precision computing machine. Digital chaos possesses the following properties: random-like behavior, ergodicity, one-way property, and sensitivity to small changes to initial conditions and control parameters. Due to these features, there has been a rise in research attempting to embed digital chaos into cryptography. Various cryptographic applications have adopted digital chaos in their designs such as image encryption [1], [2], S-box [3], [4], image watermarking [5], pseudo-random number generators [6], [7], security protocols [8], [9], and hash function [10]–[12].

Digital chaos can be classified into two groups based on the number of dimensions, one-dimension (1D) and multiple dimensions (MD). 1D chaotic maps are usually preferred for cryptographic applications because they have lower computational overhead. However, 1D chaotic maps have statistical flaws that can lead to security problems. The main issues of 1D digital maps include dynamical degradation, low complexity, and limited chaotic range. Dynamical degradation causes periodic behavior to be observed in relatively few iterations whereas low complexity allows chaotic variables (initial conditions and control parameters) to be easily estimated based on the maps' trajectories. Recovering these chaotic variables is at times analogous to recovering the encryption key of cryptographic algorithms. Also, a limited chaotic range leads to a smaller keyspace, rendering the security or encryption parameters susceptible to exhaustive search.

Many solutions have been proposed to overcome these issues and enhance 1D chaotic maps [1], [13]–[17]. One of these solutions is the deterministic chaotic finite state automata (DCFSA) [1], [13]. DCFSA has many desirable properties suitable for cryptographic applications such as a long cycle, large chaotic parameter range, high complexity,

The associate editor coordinating the review of this manuscript and approving it for publication was Tiago Cruz.

and low computational complexity. DCFSA combines existing 1D unimodal chaotic maps with finite state automata (FSA), and can form various configurations. DCFSA has the capability to produce near-ideal chaotic behavior from a 1D map without hybridizing or cascading other maps.

In 1994, the use of DNA computing in cryptography was introduced by Adleman, leading to an entirely new direction in the field [18]. DNA computing has various advantages such as high parallelism and information density, as well as low power consumption [19]. Since then, various DNA-based cryptographic algorithms have been proposed [20]–[22]. DNA encryption is made up of DNA encoding and computing, which includes biological processes and algebraic operations. DNA algebraic operations are performed on a DNA sequence. These operations include DNA addition, DNA subtraction, DNA complementary rule and DNA exclusive-OR operations [23]. Many image encryption algorithms have combined both DNA and digital chaos to leverage on the unique properties of both fields [20], [23]–[25].

Hash functions are one-way functions used to guarantee data integrity. It compresses data of any given length to a fixed-length hash value, which can then be used as a digital fingerprint for documents, messages, and images. There are many cryptographic applications that utilize hash functions, such as secret sharing [26], digital signatures [27], message authentication codes [28], and dynamic password protection [29]. Many well-known hash functions include MD4, MD5, and SHA-1 [30], which are constructed using logical operations or multi-round iteration of certain ciphers [31]. However, some standard hash functions are susceptible to differential attacks or collisions [32], [33]. To overcome these issues, newer hash functions such as SHA-3 were designed based on provably secure constructs such as the sponge construction [34]. Since its introduction, SHA-3 has been the subject of various cryptanalytic attacks [35]–[37]. Although SHA-3 is still yet to be completely broken, on-going research in the design of new and possibly more secure hash function alternatives remains vital [38].

Recently, digital chaos has been used in hash function constructions due to many similarities between the requirements of hash functions and the properties of chaotic maps. Various chaotic maps adopted in hash function algorithms include simple 1D chaotic maps [39]–[41], 2D nonlinear piecewise linear chaotic maps [42], MD chaotic maps [43], [44], coupled map lattice [45], hyperchaos [10]. Chaotic neural networks [46] and chaos-based S-Boxes [47] have also been used in hash function designs. Most chaotic hash functions use perturbation methods such as chaotic point perturbation [43], initial condition perturbation [45], and control parameter perturbation [48], [49]. Fixed-point presentation has also been explored as a means of reducing computational complexity [41]. Most of these chaotic hash functions utilize existing chaotic maps without enhancement of their chaotic performance [39]–[41], [45], [49], [50]. However, due to deficiencies in the underlying chaotic map or hash constructions being used, many of these proposed chaotic hash functions have drawbacks in terms of security and efficiency. These drawbacks can be generalized as follows:

- The simplicity of 1D chaotic maps can lead to security lapses in 1D chaotic map-based hash functions [51], [52].
- MD or hyperchaotic maps that are used to design hash functions lead to higher computational complexity [10], [43], [53], [54].
- Some chaos-based hash function only depend on one or two of the chaotic variables (initial conditions, control parameters, and chaotic points) rather than taking all of them into consideration [43], [45], [48], [49].
- Many chaos-based hash functions have highly complex constructions that lack proper security justification. This makes it difficult to verify security claims or may conceal certain shortcomings [54].

In this paper, a new chaotic hash function is proposed based on the sponge construction, DNA computing, and DCFSA. Firstly, a new DCFSA construction is proposed based on four machine states, and DNA-based state transition rule. We used four nucleotides A, T, C, and G in the design of the DCFSA's state transitions. Each state is linked to the other remaining states by one of nucleotides, and each state is associated with a logistic map with varying initial parameter values. Evaluation of its chaotic performance shows that the new DCFSA construction has a large chaotic parameter range and higher chaotic complexity as compared to its underlying logistic map. Moreover, the new DCFSA map uses four classical chaotification methods (cascade and three types of perturbations) as compared to the original DCFSA maps that used the binary chaotification methods in each configuration.

A novel chaos-based sponge construction is then proposed based on the new DCFSA configuration and DNA rules to address the security issues of conventional and chaotic hash functions. A message with arbitrary length is first padded to avoid collisions and length extension attacks. The padded message is converted into a DNA sequence based on one of the DNA rules. Meanwhile, the recurrence of DNA nucleotides is used to calculate the initial conditions for the new DCFSA map, which also contributes towards fulfilling the diffusion property. The message is then divided into a set of non-overlapping blocks and each block is XOR-ed with the sponge's bitrate DNA sequence, B_r by DNA-XOR operation in each iteration. Next, the entire sponge state (consisting of bitrate and capacity) are used to control the new DCFSA map to generate a new DNA sequence for the next iteration. The hash value is produced after processing all message blocks, and the final sponge state, S and updated chaotic variables are used in the squeezing phase. The capacity bits are used as the secret key for the proposed hash function. The proposed hash function can generate different hash sizes with uniform distribution. A standard set of statistical metrics were used to assess the proposed function. Results indicate that the proposed function has an excellent performance as compared to other chaotic hash functions. We can summarize the main contributions in this paper as follows

1) A new chaotic map based on DNA state transition, logistic map, DFA and classical chaotification methods
2) A novel chaotic hash function based on the sponge construction and DNA computing, which combines chaos with a provably secure cryptographic primitive

The remaining sections of this paper are as follows: Section II provides basic details about the sponge construction and DNA computing. Section III discusses the new DCFSA map and its chaotic analysis. Then, the proposed chaotic sponge hash function is detailed in Section IV, followed by its security and performance analysis in Section V. In section VI, a discussion of the results is provided, and the paper is concluded in Section VII.

II. PRELIMINARIES

A. SPONGE HASH FUNCTION

The sponge construction is a cryptographic primitive based on simple fixed-length permutations (or transformations) which can map a binary input string of variable length, Z_2^* to a binary output string of variable length. It has been used in the design of various cryptographic algorithms, ranging from hash functions to authenticated encryption algorithms. The behavior of a sponge construction is a combination of hash functions (variable input length to fixed output length) and stream ciphers (fixed input length to variable output length). The same internal permutation or transform is used repeatedly to process a fixed-length sponge state, either to mix inputs or generate outputs.

The sponge construction consists of three main components. The first one is the overall sponge state or state memory, S with b bits. This sponge state is divided into two sections known as the bitrate B_r and capacity C_p. The second component is the permutation (or transformation) function $f : \{0,1\}^b \rightarrow \{0,1\}^b$ which is used to process S. f is often a pseudo-random permutation of the 2^b bits of S. The last component is the padding rule, P which appends bits to the input message such that the padded message would have a length that is a multiple of the $|B_r|$. The sponge way has two phases: absorbing and squeezing as depicted in Figure 1. During the absorbing phase, bits from an input message are XOR-ed with B_r. The entire state, S is then processed by f, thus mixing bits from B_r with C_p. This is repeated until the entire input message is exhausted. The squeezing phase involves computing f before extracting bits from the bitrate to be used as the outputs of the sponge. The squeezing phase can be repeated until the desired variable-length outputs are obtained.

The sponge construction has been used to design a myriad of cryptographic algorithms due to its provable security properties. As the capacity bits are never directly affected by inputs nor extracted for outputs, it has a vital role in determining the security bound of the overall sponge function. The advantage of an adversary attempting to differentiate the sponge construction (based on a random transformation f) from a random oracle is upper-bounded by

$$\frac{N(N+1)}{2^{c+1}} \tag{1}$$

where N is the number of times f has been executed, and $c = |C_p|$ [55]. As such, the sponge construction has the same security margin as a random oracle, albeit limited to $2^{\frac{c}{2}}$ complexity. This allows cryptographers to choose suitable state lengths based on c to achieve their desired security margin. As a rule of thumb, c must be double the length of the claimed security margin, which for hash functions, is double of the desired hash value length.

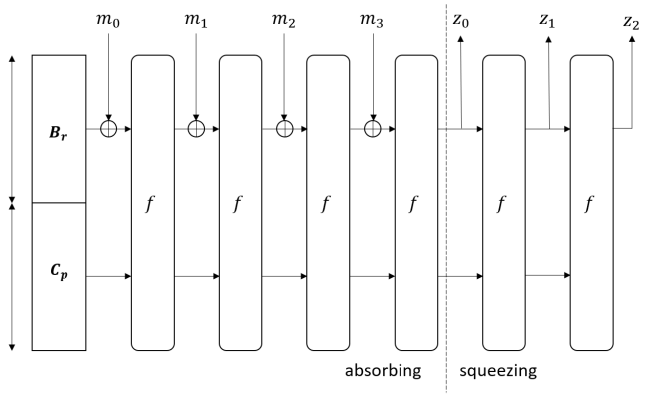

FIGURE 1. Sponge construction.

B. DNA

The DNA molecule consists of two twisted sequences, which contain four base strands: adenine (A), guanine (G), cytosine (C), and thymine (T). Under the Watson-Crick principle of pairing, A and T are complementary, and C and G are complementary. The series of nucleotides is simply a biomolecular symbolism, which can be represented by a symbolic series. DNA computing represents each nucleotide by using two bits each, {00, 01, 10, 11}. Based on the Watson-Crick rule, 8 encoding rules have been used to represent nucleotides based on these bits as summarized in Table. 1. The XOR operation between two DNA nucleotides will result in one of the four nucleotides. The mapping of the DNA-XOR operation is shown in Table 2.

TABLE 1. DNA encoding rule.

Binary	1	2	3	4	5	6	7	8
00	A	A	C	G	C	G	T	T
01	C	G	A	A	T	T	C	G
10	G	C	T	T	A	A	G	C
11	T	T	G	C	G	C	A	A

III. CHAOTIC MAP AND ANALYSIS

A. DETERMINISTIC CHAOTIC FINITE STATE AUTOMATA

In this section, we propose a new chaotic map based on DCFSA [13] and analyse its chaotic behaviours. DCFSA is a

TABLE 2. DNA-XOR rules.

XOR	A	T	G	C
A	A	T	G	C
T	T	A	C	G
G	G	C	A	T
C	C	G	T	A

new chaotification method to reduce the effect of dynamical degradation, to enhance chaotic complexity, and increase the chaotic parameter range [13] by combining DFA and 1D chaotic maps in a highly configurable manner.

In this paper, we propose a DCFSA configuration with four machine states and one chaotic map. We select the logistic chaotic map as it has been well-studied and has been used in a wide range of applications. The logistic map is defined as

$$x_{n+1} = rx_n(1 - x_n), \tag{2}$$

where $r \in [0, 4]$ is the control parameter, n is the number of iterations, and $x_n \in [0, 1]$ denotes a chaotic point, with x_0 being the initial condition.

Four classical chaotification methods (similar to the ones used in [13]) are used to design the state transition rules. Figure 2 shows the proposed DCFSA-based map with four machine sates and four state transition rules. We use DNA nucleotides (A, T, G and C) as the labels of each transition. Each state has three outgoing transitions to other states and one to itself. We associate the chaotification methods, namely classical cascade, forward perturbation, backward perturbation and parameter perturbation with A, T, G and C respectively. The state transition rule, δ in Figure 2 can be summarised as:

- $Q_1 \times A \rightarrow Q_2$
- $Q_1 \times T \rightarrow Q_3$
- $Q_1 \times C \rightarrow Q_4$
- $Q_1 \times G \rightarrow Q_1$
- $Q_2 \times A \rightarrow Q_2$
- $Q_2 \times T \rightarrow Q_3$
- $Q_2 \times C \rightarrow Q_4$
- $Q_2 \times G \rightarrow Q_1$
- $Q_3 \times A \rightarrow Q_2$
- $Q_3 \times T \rightarrow Q_3$
- $Q_3 \times C \rightarrow Q_4$
- $Q_3 \times G \rightarrow Q_1$
- $Q_4 \times A \rightarrow Q_2$
- $Q_4 \times T \rightarrow Q_3$
- $Q_4 \times C \rightarrow Q_4$
- $Q_4 \times G \rightarrow Q_1$

where Q_i are the labels of the machine state indexed by $i = \{1, 2, 3, 4\}$.

Each of the machine states of DCFSA each has a state buffer which we refer to as B_i. B_i is updated after iterating a chaotic map associated to the current state, just before transitioning to the next state. We define B_{i-1} as the buffer value from a prior state that has just transitioned to the current

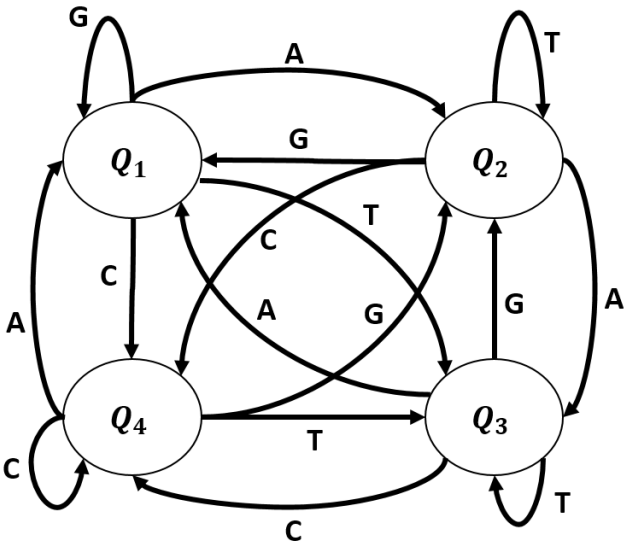

FIGURE 2. DCFSA with four machine states.

state. B_{i-1} plays an important role in the four chaotification methods that are defined as follows:

- **Cascade:** B_{i-1} is used as the initial value to iterate the chaotic map of the current state. The resulting chaotic point is updated into B_i. The cascade operation can be formally written as

$$x_{n+1} = F(c_i, r_i, B_{i-1}) \tag{3}$$
$$B_i = x_{n+1}, \tag{4}$$

  where x_n is a chaotic point at time n, F is a chaotic map, r_i is the control parameter.
- **Forward perturbation:** The chaotic map of the current state is iterated by using B_i as the initial value. The resulting chaotic point is perturbed by B_{i-1} prior to updating B_i. Forward perturbation can be mathematically defined as

$$x_{n+1} = (F(c_i, r_i, B_i) + B_{i-1}) \; mod \; 1 \tag{5}$$
$$B_i = x_{n+1}, \tag{6}$$

  where *mod* is the modular operation.
- **Backward perturbation:** Prior to iterating the chaotic map of the current state, B_i is first perturbed by B_{i-1} before being used as the initial value. The resulting chaotic point is used to update B_i. Backward perturbation can be formally written as

$$x_{n+1} = F(c_i, r_i, ((B_i + B_{i-1}) \; mod \; 1) \tag{7}$$
$$B_i = x_{n+1}. \tag{8}$$

- **Parameter perturbation:** Prior to iterating the chaotic map of the current state, the chaotic map's control parameter is first perturbed by B_{i-1}. The chaotic map is then iterated using B_i as the initial condition, whereby the resulting chaotic point is used to update B_i. The perturbed control parameter will remain as is until the next

time the machine state is visited. Parameter perturbation is mathematically defined as

$$r_i = ((r_i + (1 - B_{i-1})\zeta))\ mod\ (\zeta) + r_{min} \quad (9)$$

$$x_{n+1} = F(c_i, r_i, B_i) \quad (10)$$

$$B_i = x_{n+1} \quad (11)$$

where r_{max} and r_{min} are the chaotic range of c_i, where $\zeta = b_{max} - b_{min}$.

In each iteration of the overall DCFSA map, the resulting chaotic points are used to select the next machine state as well as the type of the chaotification method. The selection is based on a quantization function G_F. G_F is a piecewise function that divides the phase space into four. Each sub-divided region represents one of the DNA nucleotides. G_F is defined as follows

$$G_F = \begin{cases} 'G' & if\ \ x_n \leq 0.25 \\ 'C' & if\ \ x_n \leq 0.5 \\ 'T' & if\ \ x_n \leq 0.75 \\ 'A' & if\ \ x_n \leq 1. \end{cases} \quad (12)$$

The advantages of the new DCFSA map over the one proposed in [13] include:

- A larger number of machine states (4 rather than 3 in [13]).
- The state transition rule in [13] consists of threshold values whereas the proposed DCFSA is consists of four state transition rules (DNA rules).
- In [13], we used two chaotification methods for each DCFSA model. In contrast, here we used four chaotification methods.
- We used four DNA nucleotides as labels of each transition whereas in [13], two binary labels were used.

B. DCFSA ANALYSIS

The new map depicts highly chaotic behavior and high sensitivity to minute changes to its variables (initial conditions and control parameters). This is in addition to an extended chaotic range and enhanced complexity as compared to its underlying logistic map. To investigate these properties, we use bifurcation diagrams, LE and fuzzy entropy (FuzzyEn). The bifurcation diagram is commonly used as an indicator of the chaotic parameter range while also providing a clear view of chaotic behaviors. Figure 3 shows the chaotic range of the new DCFSA map and logistic map. The chaotic map for each of the DCFSA's states has a control parameter that is varied within the range of $r \in (0, 4]$ while using the same initial condition $x_i = 0.137$, where $i = \{1, 2, 3, 4\}$. We can see that the new DCFSA map has a large chaotic range as compared to the logistic map. Periodic windows that exist within the chaotic range of the logistic map has also been removed.

LE is used to quantify the sensitivity of chaotic maps. It measures the rate of divergence of a dynamical system when starting from two infinitesimally close initial conditions. Large positive LE values are desired in chaos-based

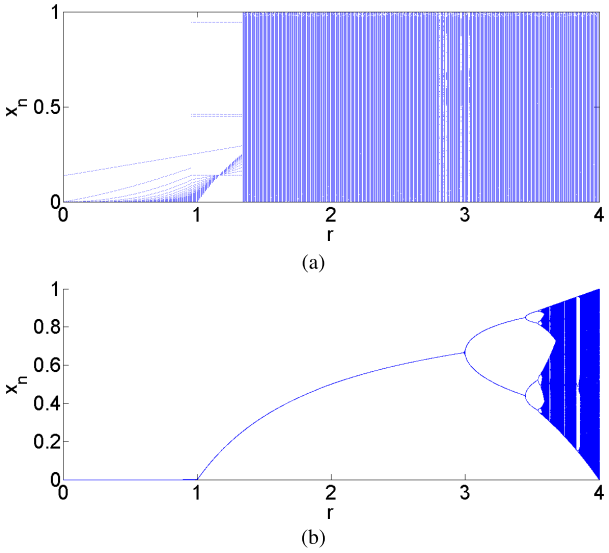

FIGURE 3. Bifurcation diagram of (a) DCFSA map (b) logistic map.

applications. Large LE values imply high sensitivity and rate of divergence. The DCFSA map has a unique property whereby the divergence of chaotic points influences state transitions. Thus, a large LE value will lead to highly unpredictable transitions, which then leads to a greater divergence between chaotic points. To study this property, we first evaluate the sensitivity of the machine states rather than the chaotic points. Figure 4 shows the machine state sensitivity as a function of its control parameter value. We can observe that only a few iterations are required for two DCFSA maps starting from two initial conditions, $x_0 = 0.13,\ y_0 = 0.13 + 10^{-15}$ to diverge in terms of state transitions. Next, we compare the LE values of the DCFSA map against its underlying logistic map in Figure 5. One can see that the new DCFSA map has larger positive values as compared to the logistic map. Both of these results depict the sensitivity of the DCFSA map to small changes in its initial conditions and control parameters, making it more suitable than the logistic map for cryptographic applications.

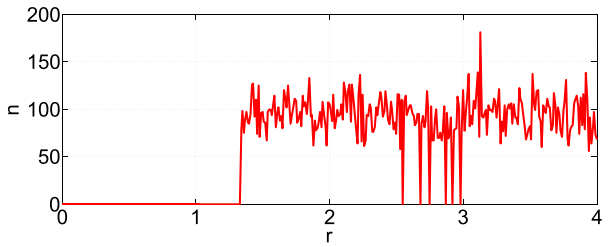

FIGURE 4. State machine sensitivity, two chaotic trajectories are started from two closed initial points 0.13, 0.130000000000001, n is number of iterations to different machine states.

FuzzyEn is numerical measurement of complexity and randomness. If a chaotic system does not have identifiable patterns in its orbit, it is considered to be highly complex, with parameters that are hard to estimate. This property is

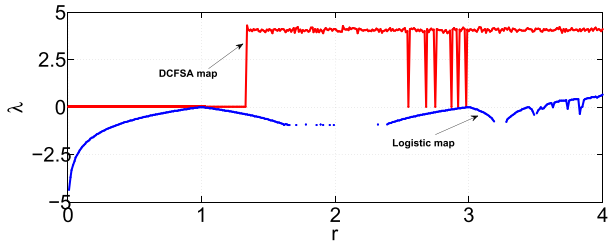

FIGURE 5. LE values of DCFSA map and logistic map.

especially vital for cryptographic algorithms based on chaotic maps to provide security against parameter estimation and brute force attacks. As such, a high FuzzyEn value is desired for chaos-based cryptographic algorithms. Figure 6 shows that the new DCFSA map has highly complex and irregular behavior as compared to the logistic map.

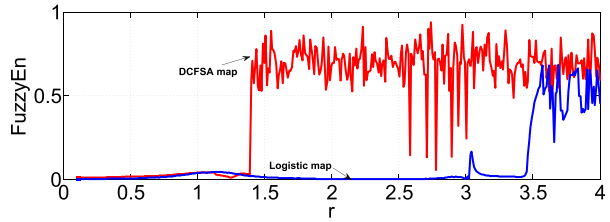

FIGURE 6. FuzzyEn values of DCFSA map and logistic map.

IV. PROPOSED HASH FUNCTION

The proposed hash function is based on the sponge construction, DNA computing and DCFSA map. To the best of the authors' knowledge, this is the first time that all three fields have been unified for cryptographic application. The hash function can be divided into three main phases: message pre-processing, absorbing and squeezing. The message pre-processing phase includes steps such as DNA transform, message padding, sponge state initialization and initial condition extraction. The absorbing phase consist of two operations: DNA-XOR operation (bitrate B_r will be XOR-ed with a message block) and DCFSA iterations based on the updated sponge state. The DNA-XOR and DCFSA iterations are part of the compression function, γ. The absorbing phase is repeated until all message blocks have been compressed. This results in an updated sponge state, modified control parameters and DCFSA buffer values, all of which have been influenced by the entire input message. Finally in the squeezing phase, additional γ iterations are performed before the final hash value is extracted from the bitrate, B_r. Figure 7 depicts the entire hashing process. In the following subsections, we delve into the details of each phase and sub-phases.

A. MESSAGE PRE-PROCESSING

The input message M is first padded to be a multiple of 128 bits (to match the size of B_r of the sponge state). To deter collisions and length extension attacks, we use a sponge-compliant padding rule that consists of a single 1 followed by 0 bits (e.g. 1000*). We use the last 32 bits for storing length of input message. For example, (0000 . . . 1001110001000) represents a message length of 5000 bits. Next, DNA transform converts the input message into a DNA sequence, by converting pairs of bits into one of DNA nucleotides ($11 \rightarrow A$, $00 \rightarrow T$, $01 \rightarrow C$ and $10 \rightarrow G$) For this, we use rule number 7 from Table 1. The DCFSA has four machine states that requires four initial conditions. Thus during transformation process, the number of each DNA nucleotide is counted and used to initialize the initial conditions. Let h_1, h_2, h_3 and h_4 be the number of each DNA nucleotide A, T, C, and G, respectively. The initial conditions are then calculated as

$$x^0{}_1 = ((h_2 + h_3 + h_4) + (h_1 \times 9^7)) \bmod 0.999, \quad (13)$$
$$x^0{}_2 = ((h_1 + h_3 + h_4) + (h_2 \times 9^7)) \bmod 0.999, \quad (14)$$
$$x^0{}_3 = ((h_2 + h_1 + h_4) + (h_3 \times 9^7)) \bmod 0.999, \quad (15)$$
$$x^0{}_4 = ((h_2 + h_3 + h_1) + (h_4 \times 9^7)) \bmod 0.999, \quad (16)$$

where $sum(h_i)$ is total number of all DNA nucleates in the DNA sequence. For example, A is counted 1034 in an input message. $sum(h_i)$ is used to create initial conditions that are high sensitive to a small change, and that lead to diffusion and confusion properties in the hash value. The constant 9^7 is used to increase the effect of small changes in the input message on the resulting initial conditions. The 384-bit $(128 + 256)$ sponge state is then initialized by using the first 128 bits (64 DNA nucleates) from the input message as B_r and the 256-bit (128 DNA nucleates) secret key as C_p. Finally, the input message is divided into 128-bit (64 DNA nucleates) blocks. The entire message pre-processing phase is as shown in Algorithm 1.

B. ABSORBING PHASE

The absorbing phase plays the most important role in the proposed hash function: compressing the entire input message. The compression function, γ relies on DNA-XOR operations and iterations of the DCFSA map to process all blocks of the input message. The values of C_p and B_r are used to control the DCFSA map to generate a new DNA sequence equivalent to the size of the entire sponge state $B_r + C_p$. Chaotic points generated by the DCFSA map are converted to DNA nucleates that are used to directly update the sponge state. The modified DCFSA's buffer values and control parameters will be retained and used to process the next message block. Subsequent message blocks are DNA-XORed with B_r prior to executing γ. The process is repeated for all message blocks, resulting in a sponge state and DCFSA parameters that are dependent on the entire input message and secret key. Unlike other chaos-based hash functions that are dependent on either the chaotic points or control parameter, the proposed hash function is affected by both the DCFSA's chaotic points and control parameters that are dynamically modified simultaneously in each iteration. Figure 8 visually depicts the absorbing phase whereas Algorithms 2 and 3 describes it in detail. In Algorithm 3, *Seq* is used to control the DCFSA map, and the generated chaotic points are converted to DNA

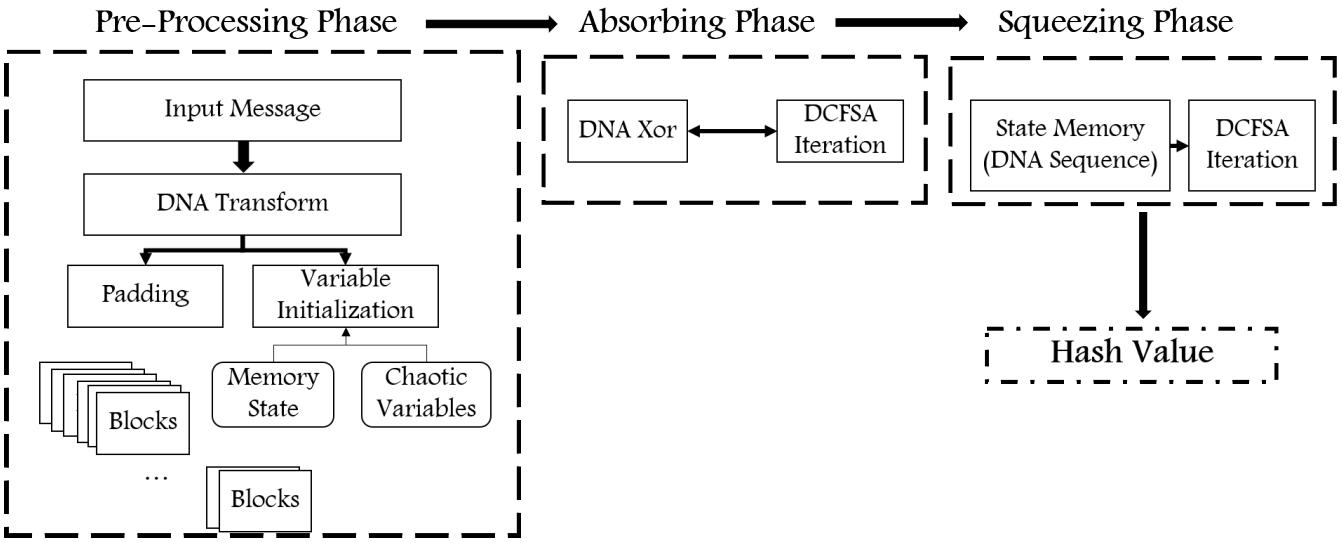

FIGURE 7. Phases of the proposed hash function.

nucleates. These generated DNA nucleates are stored in the sponge state memory (B_r and C_p). The new state memory values and chaotic variables are used for the next iteration of γ.

C. SQUEEZING PHASE

This phase is responsible to generate the resulting hash value using the sponge state that has been modified by the absorbing phase. The hash value is produced by processing γ repeatedly and extracting 128-bit sequences from B_r as the hash value. Each generated chaotic point is converted into one of the DNA nucleates, each of which is 2 bits. For hash sizes other than 128-bit, the squeezing phase is performed repeatedly.

The chaotic points are converted to DNA nucleates based on the following: $x_i \leq 0.25$, $Seq(i) \rightarrow G$, $x_i \leq 0.5$, $Seq(i) \rightarrow C$, $x_i \leq 0.75$, $Seq(i) \rightarrow T$, and $x_i \leq 1$, $Seq(i) \rightarrow A$. Each DNA nucleate is used to control the transition between machine states (which also determines the chaotification method being applied). This decision impacts the generation of the next chaotic point, which is then converted into a new DNA nucleate. Figure 9 provides a visual depiction of the squeezing phase.

V. EXPERIMENTAL EVALUATION

We evaluate the proposed hash function in terms of distribution, sensitivity, diffusion, confusion, and collision analysis. We use the same set of experiments that has been used to evaluate the current state-of-the-art in chaos-based hash functions [10], [12], [41], [53], [56]. The control parameters that are used in these experiments are $r_i = 3.99$. Unless stated otherwise, we use input messages that consist entirely of the letter a in all experiments. All the experiments are coded and executed on Matlab using a computer with Intel Core i5-560M @ 2.67GHz Processor, 8GB RAM, and Windows 7 operating system.

A. DISTRIBUTION OF HASH VALUE

A hash function should generate uniformly distributed hash values for different input messages. Without uniform distribution, there may be information leakage that can be used by an adversary to identify collisions or perform forgery attacks. To evaluate the proposed hash function's distribution, we randomly selected 1,000 input messages with 2^{11} characters for hashing. The hash values of these messages are transformed into hexadecimal values to study their distribution. The number of occurrences for each hexadecimal value is counted for all input messages, resulting in the uniform distribution shown in Figure 10. To further validate the uniform distribution of the proposed hash function, we hashed $N = 10,000$ different input messages and count the number of toggled bits at each bit position. Theoretically for an ideal distribution, the bits have to change $\frac{N}{2} = 5,000$ times. The proposed hash function achieves a mean value of $5,042$, which is near-ideal. Figure 11 shows the distribution test results for $N = 10,000$. We can observe that the all bit locations are close to the theoretical value, which indicates that the proposed hash function has ability to resist the statistical attacks.

Linear complexity is another metric to evaluate the differences between 1s and 0s in the output of a hash function. To study the linear complexity, we count the number of 0s in the hash value for different input messages. It is desirable for the number of 1s to be equal to 0s. Figure 12 depicts the linear complexity of the proposed hash function for N different messages. We can see the proposed hash function has near-ideal linear complexity.

B. SENSITIVITY TEST

A secure hash function must be highly sensitive to the message or secret key. The initial conditions and control parameters of the underlying DCFSA map plays a critical role in

Algorithm 1 Pre-Processing Message

Data: Input message M.
Result: $x_j, j = 1, 2, 3, 4$ initial conditions, m_i message blocks

$N = length(M)$;
$remainder = N \ mod \ 128$;
$LenBin = dec2bin(N, 32)$;
if $remainder == 0$ **then**
  $M(N+1) =' 1'$;
  $M(N+2 : N+96) =' 0'$;
  $M(N+97 : N+128) = LenBin(1 : 32)$;
else
  if $remainder < 32$ **then**
    if $remainder > 1$ **then**
      $M(N+1) =' 1'$;
      $M(N+2 : N+remainder) =' 0'$;
    else
      $M(N+1) =' 1'$;
    $N = N + remainder$;
    $M(N+1) =' 1'$;
    $M(N+2 : N+96) =' 0'$;
    $M(N+97 : N+128) = LenBin(1 : 32)$;
  else
    $Z = remainder - 32$;
    if $Z == 0$ **then**
      $M(N+97 : N+128) = LenBin(1 : 32)$;
    else
      if $Z == 1$ **then**
        $M(N+1) =' 1'$;
        $M(N+97 : N+128) = LenBin(1 : 32)$;
      else
        $M(N+1) =' 1'$;
        $M(N+2 : N+Z-33) =' 0'$;
        $M(N+97 : N+128) = LenBin(1 : 32)$;

Convert the M into DNA nucleates;
Calculate number of A, T, C and G ;
Calculate the initial conditions $x_j, j = 1, 2, 3, 4$ based on the Eq.13,14,15, and 16, respectively;
Divided M into non-overlapping blocks m_i

Algorithm 2 Absorbing Phase

Data: Input message m_i, where N is total number of blocks. B_r and C_p are initialized. Initial conditions x_j and control parameter $r_j = 3.9, j = 1, 2, 3, 4$.
Result: B'_r and C'_p and $C_p{}'$, x'_j, and r'_j

for $i = 1$ *to* N **do**
  $B_r = DNA_{XOR}(B_r, m_i)$;
  $C_p = DCFSA(C_p, X_j, r_j)$;
  $B_r = DCFSA(B_r, X_j, r_j)$;

the resulting hash values. As such, it is also important to consider the sensitivity of these parameters. In this test, an input message is used with 2^{11} $'a'$ characters. Then, we apply a

Algorithm 3 DCFSA Map Iteration

Data: DNA sequence, Seq. Initial conditions x_j and control parameter $r_j = 3.99, j = 1, 2, 3, 4$.
Result: Seq, x_j, and r_j $Q = 1$ $B = x_1$;
for i *= 1 to length(Seq)* **do**
  if *Seq(i)=='A and Q==1* **then**
    $x_1 = mod(r_1 * x_1 * (1 - x_1) + B, 1)$; $Q = 2$; $B = x_1$
  else **if** *Seq(i)=='T and Q==1* **then**
    $x_1 = mod(B + x_1, 1)$; $x_1 = r_1 * x_1 * (1 - x_1)$; $Q = 3$; $B = x_1$;
  else **if** *Seq(i)=='C' Q==1* **then**
    $r_1 = (1 - B) * 0.4 + 3.6$; $x_1 = r_1 * x_1 * (1 - x_1)$; $Q = 4$; $B = x_1$;
  else **if** *Seq(i)=='G' Q==1* **then**
    $x_1 = r_1 * B * (1 - B)$; $Q = 1$; $B = x_1$;
  else **if** *Seq(i)=='A' Q==2* **then**
    $x_2 = mod(r_2 * x_2 * (1 - x_2) + B, 1)$; $Q = 2$; $B = x_2$;
  else **if** *Seq(i)=='T' Q==2* **then**
    $x_2 = mod(B + x_2, 1)$; $x_2 = r_2 * x_2 * (1 - x_2)$; $Q = 3$; $B = x_2$;
  else **if** *Seq(i)=='C' Q==2* **then**
    $r_2 = (1 - B) * 0.4 + 3.6$; $x_2 = r_2 * x_2 * (1 - x_2)$; $Q = 4$; $B = x_2$;
  else **if** *Seq(i)=='G' Q==2* **then**
    $x_2 = r_2 * B * (1 - B)$; $Q = 1$; $B = x_2$;
  else **if** *Seq(i)=='A' Q==3* **then**
    $x_3 = mod(r_3 * x_3 * (1 - x_3) + B, 1)$; $Q = 2$; $B = x_3$;
  else **if** *Seq(i)=='T' Q==3* **then**
    $x_3 = mod(B + x_3, 1)$; $x_3 = r_3 * x_3 * (1 - x_3)$; $Q = 3$; $B = x_3$;
  else **if** *Seq(i)=='C' Q==3* **then**
    $r_3 = (1 - B) * 0.4 + 3.6$; $x_3 = r_3 * x_3 * (1 - x_3)$; $Q = 4$; $B = x_3$
  ; else **if** *Seq(i)=='G' Q==3* **then**
    $x_3 = r_3 * B * (1 - B)$; $Q = 1$; $B = x_3$;
  else **if** *Seq(i)=='A' Q==4* **then**
    $x_4 = mod(r_4 * x_4 * (1 - x_4) + B, 1)$; $Q = 2$; $B = x_4$;
  else **if** *Seq(i)=='T' Q==4* **then**
    $x_4 = mod(B + x_4, 1)$; $x_4 = r_4 * x_4 * (1 - x_4)$; $Q = 3$; $B = x_4$;
  else **if** $Seq(i) ==' C' \ Q == 4$ **then**
    $r_4 = (1 - B) * 0.4 + 3.6$; $x_4 = r_4 * x_4 * (1 - x_4)$; $Q = 4$; $B = x_4$;
  else;
  –
  $x_4 = r_4 * B * (1 - B)$; $Q = 1$; $B = x_4$;
  if $B \leq 0.25$ **then**
    $Seq(i) =$ 'G';
  else **if** $B \leq 0.5$ **then**
    $Seq(i) =$ 'C';
  else **if** $B \leq 0.75$ **then**
    $Seq(i) =$ 'T';
  else;
  $Seq(i) =$ 'A';

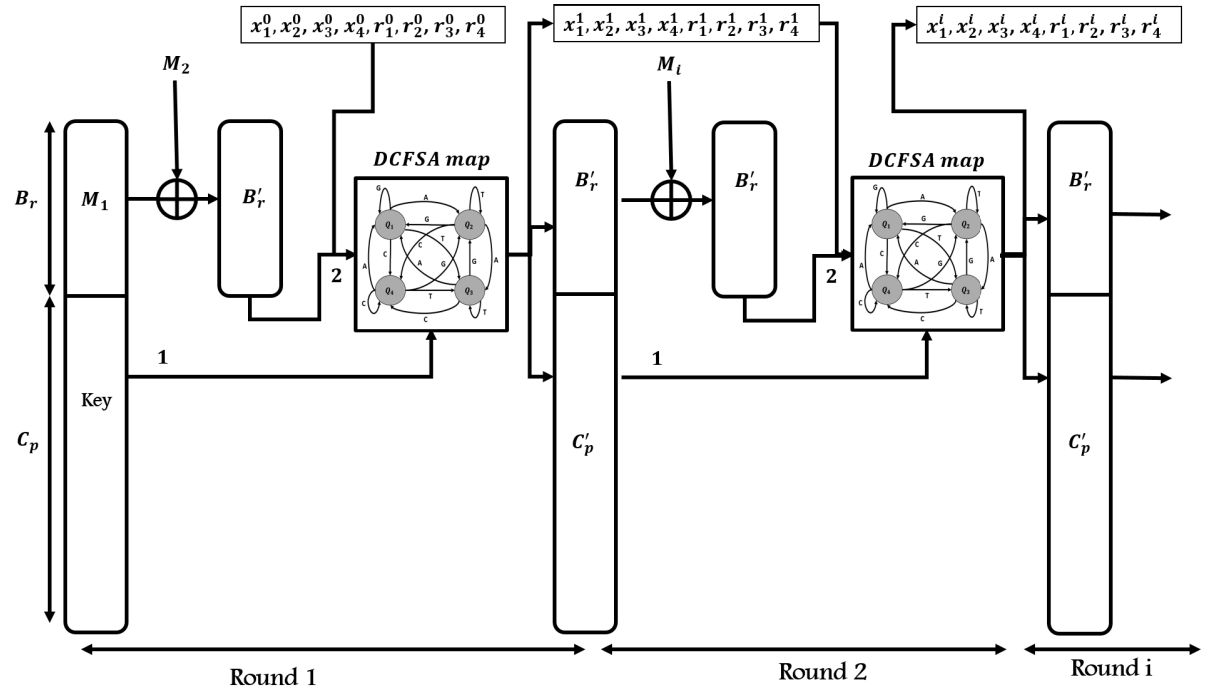

FIGURE 8. Absorbing Phase of the proposed hash function.

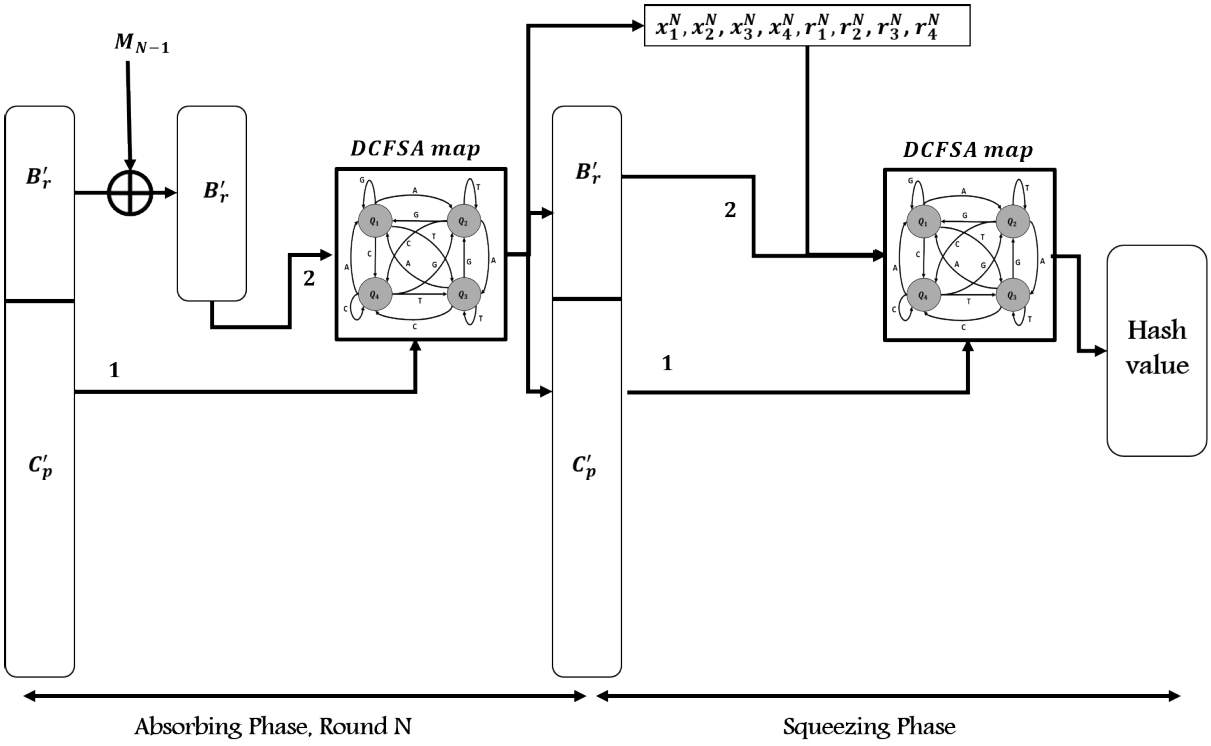

FIGURE 9. Squeezing Phase of the proposed hash function.

set of small modifications to the input message and chaotic variables, for which hash values are generated. These modifications include:

Case 1 : An input message M consisting of 2^{11} $'a'$ characters

Case 2 : Flip the first bit of M

Case 3 : Flip the last bit of M

Case 4 : Flip the middle bit of M

Case 5 : Small change to control parameter in the machine state Q_1 $r_1 + 10^{-15}$

Case 6 : Small change to control parameter in the machine state Q_2 $r_2 + 10^{-15}$

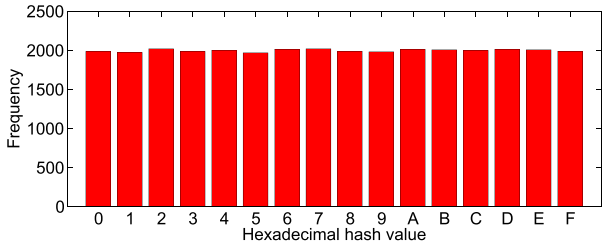

FIGURE 10. Distribution hexadecimal hash value for 1, 000 different input messages.

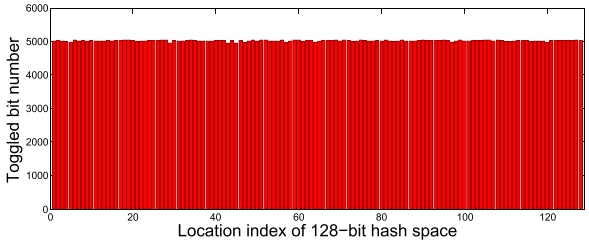

FIGURE 11. Distribution of hash value in hash space for 10, 000 different input messages (Theoretical value = 5,000).

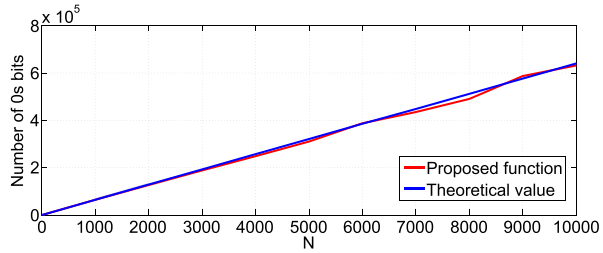

FIGURE 12. Linear complexity of hash value for *N* different input messages.

Case 7 : Small change to control parameter in the machine state Q_3 $r_3 + 10^{-15}$

Case 8 : Small change to control parameter in the machine state Q_4 $r_4 + 10^{-15}$

Case 9 : Flip the last bit of the capacity C_p

A binary representation of the six hash resulting hash values are shown in Figure 13. Despite the minute changes that have been performed, the resulting hash values are entirely different, showing a high degree of sensitivity to small changes of the input message and chaotic variables.

Table 3 contains a hexadecimal representation of nine hash values for varying cases. The ratio of bits changed for each case as opposed to Case 1 is listed in the table, demonstrating the hash function's high sensitivity to its inputs and avalanche qualities.

The DNA-based DCFSA has a strong level of security as shown in section III-B. In the proposed hash function, the whole input message and the secret key have a significant effect on the output of the hash function. Both parameters that modify the sponge state and the DCFSA variables. A tiny change to the input message or secret key will incur small changes to the nucleates, machine states, chaotification methods and/or initial conditions. Thus, changing a single bit has

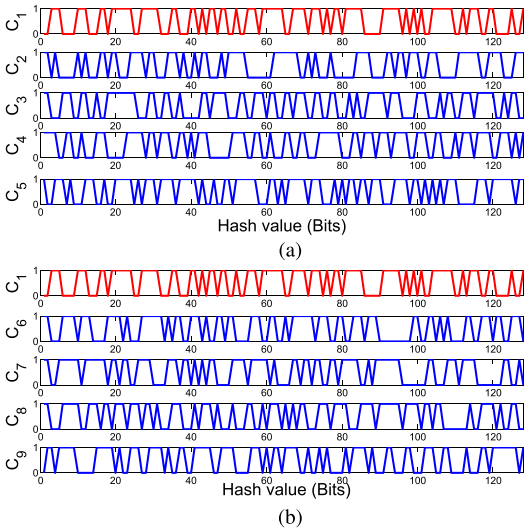

FIGURE 13. 128 bits hash values in different nine cases (a) case 1 against the cases from two to five (b) case 1 against the cases from six to nine.

a strong avalanche effect as these small modifications will lead to major changes in the resulting hash value. In addition, the chaotic points and control parameters of DCFSA in the proposed hash function are dynamically changed simultaneously in each iteration.

C. DIFFUSION AND CONFUSION TEST

Diffusion and confusion are important properties for a hash function. To analyze the proposed hash function for both of these properties, the following steps are followed:

1) An input message M is selected and hashed to obtain H_1.
2) Change one random bit in M to obtain M'.
3) Hash M' to obtain H_2.
4) Compare the number of bits changed, B between H_1 and H_2.
5) Repeat the test N times.

B is then used to calculate the following metrics that allow us to quantify diffusion and confusion:

-Minimum changed bit number $B_{min} = \min(B_i)_1^N$

-Maximum changed bit number $B_{max} = \max(B_i)_1^N$

-Mean changed bit number $\bar{B} = \sum_1^N \frac{B_i}{N}$

-Mean changed probability $P = \frac{\bar{B}}{128} \times 100\%$

-Standard variance of the changed bit number $\triangle B = \sqrt{\frac{1}{N-1}\sum_1^N (B_i - \bar{B})^2}$

-Standard variance of probability $\triangle P = \sqrt{\frac{1}{N-1}\sum_1^N (\frac{B_i}{128} - P)^2} \times 100\%$

The test results for $N = 512, 1024, 2048$ and 10000 are tabulated in Table 4. For an ideal hash function, $\bar{B}$ and P should have values of 64 and 50% respectively as well as low values of $\triangle B$ and $\triangle P$. For visual analysis, Figure 14 shows the frequency of the average changed bit numbers. We can see that $\bar{B}$ is near to the perfect value of 64. Moreover, the standard variance values are low, indicative of robust diffusion and confusion properties. In Table 5, we benchmark

TABLE 3. 128 bits hash values in different six cases and percentage of number of bits changed.

Case	Hash value	Ratio of number of bits changed
Case 1	88953202f2b6ab35fd181f60c6f554ad	
Case 2	0b26d26f046cc001ea6f5d8c5f6de98a	49.51
Case 3	7801d6e9a9c6b6d9afff98012e3daf3b	51.35
Case 4	98256fcd66fe9da98fe2de56601265fd	50.82
Case 5	8ac6914fe6218da698ca66624dea6f6f	50.11
Case 6	f79586592a6f5d8c544ea68fc1c1ba45	51.54
Case 7	a45d56ec69d8a9650102ca336e6d6ff6	51.95
Case 8	8982dc6e61a7a334889256fda5123bc5	49.25
Case 9	7845129e53ac5c5b6ae6d9f9ff5acc65	49.51

TABLE 4. Statistical results for $N = 512$, 1024, 2048, 10,000 and 128-bit hash.

Parameters	Proposed function			
	512	1024	2048	10,000
$\bar{B}$	63.892	63.914	63.941	63.922
$P\%$	49.784	49.762	49.832	50.089
$\triangle B$	5.666	5.768	5.689	5.688
$\triangle P$	4.497	4.418	4.413	4.324
B_{min}	49	46	44	42
B_{max}	82	82	82	84

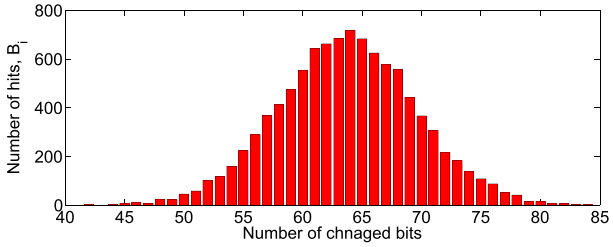

FIGURE 14. Distribution of changed bit number B_i and its histogram.

TABLE 5. Comparison of diffusion and confusion characteristics $N = 2048$.

Algorithm	$\bar{B}$	$P(\%)$	$\triangle B$	$\triangle P$
Proposed function	63.941	49.83	5.69	4.41
Ref. [57]	63.94	49.95	5.61	4.38
Ref. [48]	64.27	50.21	5.59	4.36
Ref. [53]	63.87	49.90	5.58	4.36
Ref. [50]	64.10	50.08	5.58	4.36
Ref. [58]	64.12	50.09	5.63	4.41
Ref. [47]	63.99	49.99	5.66	4.40
Ref. [56]	64.01	50.01	5.66	4.26
Ref. [59]	62.65	48.94	5.48	4.28
Ref. [44]	63.94	49.95	5.69	4.44
Ref. [43]	63.89	49.92	5.77	4.51
Ref. [46] (Structure 1)	64.05	50.03	5.65	4.41
Ref. [46] (Structure 1)	63.88	49.91	5.66	4.42
Ref. [46] (Structure 1)	63.90	49.92	5.60	4.37
Ref. [60]	64.07	50.06	5.74	4.48
Ref. [61]	64.17	50.14	5.78	4.51
Ref. [62]	64.43	50.34	5.99	4.68
Ref. [63] MD5	64.03	50.02	5.66	4.42

our proposed hash function against the current state-of-the-art. In addition to near-ideal confusion and diffusion properties, it is at least on par with or outperforms other chaos-based hash functions.

D. COLLISION ANALYSIS

A collision attacks aims to generate to equivalent hash values from two different inputs. As a basic requirement, all hash functions should be robust against collision attacks. To analyze the proposed hash function's collision resistance, we first use two input messages with one bit of difference, and calculate their corresponding hash values. The two hash values are represented as ASCII characters for comparison purposes. Let *hit* denote the number of ASCII characters in the same location in the hash value that are equal. Let ω be the number of hits $\omega = 0, 1, 2, \ldots, s$. $W_N(\omega)$ refers to the number of hits that occur for a test with N samples. The theoretical $W_N(\omega)$ values for distinct N is computed as

$$F = \begin{cases} W_N(\omega) = N \times \dfrac{s!}{\omega!(s-\omega)!} \times (\dfrac{1}{2^8})^{\omega} \times (1 - \dfrac{1}{2^8})^{s-\omega} \\ \displaystyle\sum_{\omega=0}^{s} W_N(\omega) = W_N(0) + W_N(1) + \ldots + W_N(s) = N \end{cases} \tag{17}$$

where $s = \frac{128}{8} = 16$ (128 is the length of the hash value whereas 8 bits are required for ASCII representation). When $N = 10,000$ the theoretical values are $W_N(0) = 9392.98$, $W_N(1) = 589.36$, $W_N(2) = 17.33$, $W_N(3) = 0.3172, \ldots, W_N(16) = 2.94 \times 10^{-35}$, and so on.

We perform the test on the proposed hash function, which achieves values of $W_N(0) = 9365$, $W_N(1) = 600$, $W_N(2) = 35$ and $W_N(> 2) = 0$. These results are visually depicted in Figure 15. The collision resistance of the proposed hash function closely resembles the ideal case. Furthermore, Table 6 provides a comparison with the collision resistance of other chaos-based hash functions. These results imply that the proposed hash function is highly resistant to collision attacks.

E. KEYSPACE ANALYSIS

The keyspace of a cryptographic algorithm should be sufficiently large to resist an exhaustive key search. Minimally, the keyspace of a cryptographic algorithm should be at least 2^{128}. The proposed hash function has the capability to function either as a keyed or unkeyed hash function. When in keyed mode, a 256-bit secret key is used to initialize the capacity, C_p of the sponge state. Rather than using the secret key bits to initialize chaotic parameters, it will be used to dictate the behavior of the underlying DCFSA map. This

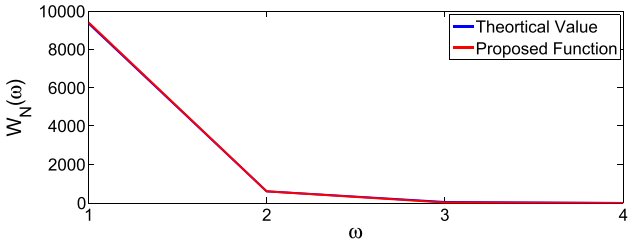

FIGURE 15. Distribution of the number of equal ASCII values at the same location in hash values.

TABLE 6. Number of hits for $N = 10000$.

$N = 10000$							
Hits	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Ideal	9393	589	17	0	0	0	0
Proposed function	9365	600	35	0	0	0	0
Ref. [48]	9554	404	42	0	0	0	0
Ref. [53]	9387	586	27	0	0	0	0
Ref. [47]	9401	580	19	0	0	0	0
Ref. [56]	9969	31	0	0	0	0	0
Ref. [44]	9431	557	12	0	0	0	0
Ref. [43]	9387	591	22	0	0	0	0
Ref. [64]	9359	626	15	0	0	0	0
Ref. [57]	9414	569	17	0	0	0	0

provides an advantage over other chaos-based algorithms that use the secret key to initialize chaotic parameters because this leads to weak key problems [54]. The keyspace of 256 bits is more than sufficient for the proposed hash function to resist brute force attacks. Furthermore, Table 7 shows the comparison with other hash functions [41], [48], [65]–[67]. One can see the keyspace is very large compare as compared to the other hash functions.

TABLE 7. Comparison of keyspace values.

Algorithm	Keyspace
Proposed function	2^{256}
Ref. [10]	$2^{383.9}$
Ref. [41]	2^{128}
Ref. [48]	2^{106}
Ref. [65]	2^{192}
Ref. [66]	$> 2^{128}$
Ref. [67]	2^{199}

F. RESISTANCE TO BIRTHDAY ATTACK

A hash function should be resistant against the birthday attack, whereby an attacker would only need to test 50% of all possible hash values to find a collision. For a hash value with length n, an attacker can test $2^{\frac{n}{2}}$ possibilities to identify a collision. The proposed hash function has a hash value length of 128 bits. To find a collision, an adversary would require at least 2^{64} guesses. Based on the current computing capability, 2^{64} operations is still impractical or costly to compute. With that said, it is easy to extend the hash length by computing multiple iterations of γ to generate a larger hash value if need.

G. ABSOLUTE DIFFERENCE

In this test, an input message and its corresponding hash value are generated, similar to the experiment in Section V-D. The acquired hash values are converted to ASCII for comparison purposes. We can calculate the absolute difference d as

$$d = \sum_{i=1}^{16} |t(e_i) - t(e_i')| \tag{18}$$

where $t()$ is a function for converting an ASCII character into its corresponding decimal value while e_i and e_i' are the two ASCII hash values for both original and altered input messages respectively. The ideal mean value of the absolute difference for two hash values is equal to 1, 360 for a 128-bit hash [11], [43]. In the proposed hash function, we calculate d for $N = 10,000$ experiments and the resulting mean is 1,353.524, which is very close to the ideal value. This implies that the proposed hash values have near-ideal uniform distribution and that all the characters of the hash value have the same possibility of occurring.

H. FLEXIBILITY

The flexibility property of the proposed hash function can be summarized as follows:

- The underlying DCFSA map can be easily replaced with any other DCFSA map configurations to produce different hash functions.
- The sponge state (bitrate B_r and capacity C_p) has a flexible length. We can increase the size of the sponge state to generate larger hash values that will still fulfil the indifferentiability property described in Section II-A.
- The DNA sequence has 8 encoding rules that a user can select from to transform input messages to DNA sequences. Furthermore, these rules can also be dynamically selected during the hashing process.
- Larger hash values can be easily accommodated by just iterating the compression function, γ during the squeezing phase.

I. SPEED ANALYSIS

In the proposed hash function, each block has 128 bits, which is equivalent 64 DNA symbols. The DNA-XOR is applied in each round to 64 DNA symbols and $128 + 64 = 192$ DNA symbols use to control DCFSA to generate new 192 DNA symbols for bitrate B_r and capacity C_p. Thus, we can estimate the overall computational complexity of the proposed hash functions by taking the number of blocks B multiplied by 192 chaotic iterations and multiplied by 64 DNA-XOR operations, $B \times 192 + B \times 64$. To compare the speed of the proposed hash function with its chaos-based peers, the proposed hash function was implemented using Matlab R2012b on a Intel Core i5-560M @ 2.67GHz Processor, 8GB RAM, and Windows 7 operating system. We can observe that the proposed hash function outperforms some of the existing chaotic hash functions as shown in Table 8.

TABLE 8. Hashing speed comparison.

Algorithm	Speed (Gbps)	Platform
Proposed function	**1.234**	**Intel Core-i5, 8GB RAM**
Ref. [41]	4.04858	Intel Core-i7, 16GB RAM
Ref. [53]	0.911	Intel Core-i5, 16GB RAM
Ref. [49]	0.697	Intel Core-i7, 16GB RAM
Ref. [56]	0.666	Intel Core-i7 (4 core), 8GB RAM
Ref. [58]	0.630	Intel Core-i3, 4GB RAM
Ref. [47]	0.129	Intel Pentium IV (2 core), 2GB RAM
Ref. [44]	0.068	Intel Core-i7, 4GB RAM
Ref. [43]	0.044	Not specified
Ref. [64]	≈ 0.0019	Intel Core2, 2GB RAM
Ref. [68]	≈ 0.00015	Intel Pentium IV, 256MB RAM

J. COMPARISON WITH OTHER CHAOS-BASED ALGORITHMS

To depict the strength of the proposed hash function, we perform a comparison with other chaotic hash functions [10], [43]–[45], [47]–[49], [56]. The selected evaluation metrics include diffusion, confusion and collision analysis as they are the mandatory requirements of a secure hash function. In Table 9, we can see that the proposed hash function outperforms the rest, despite the other hash functions already having near-perfect statistical results.

TABLE 9. Performance comparison of diffusion, confusion and collision characteristics.

Algorithm	Diffusion and Confusion, N=2048	Collision, N=10000		
	$\bar{B}$	$W_N(0)$	$W_N(1)$	$W_N(2)$
Proposed function	63.94	9365	600	35
Ref. [49]	64.24	9381	581	38
Ref. [45]	64.17	9495	505	0
Ref. [10]	64.095	9396	583	21
Ref. [48]	64.27	9554	404	42
Ref. [44]	63.94	9431	557	12
Ref. [53]	63.87	9387	586	27
Ref. [47]	63.99	9401	580	19
Ref. [56]	64.01	9969	31	0
Ref. [43]	63.89	9387	591	22
Ideal value	**64**	**9393**	**589**	**17**

We now compare the proposed hash function against two well-known hash functions that are widely used today, SHA2-256 and SHA3-256. In this comparison, hash values (the proposed function, SHA2-256 and SHA3-256) are generated with the same length for a fair comparison. The diffusion, confusion, and absolute difference metrics are calculated these three hash values and tabulated in Table 10. The proposed hash function obtains near-ideal scores that are similar to both SHA hash functions.

TABLE 10. Performance comparison of diffusion, confusion and mean value of absolute difference with secure hash algorithms (SHA).

	Proposed Function	SHA2-256	SHA3-256
$\bar{B}$	127.988	127.968	128.019
$P\%$	50.043	50.084	50.043
$\triangle B$	8.106	8.108	8.079
$\triangle P$	3.113	3.107	3.113
B_{min}	99	100	155
B_{max}	156	100	155
d	2715	2642	2708

VI. DISCUSSION

In the discussion, we analyze the proposed hash function based on its three components and explore the strength and weakness of each one. First, the DNA sequence has an advantage in reducing number of bits. Most hash functions divide input messages into blocks which are then converted into a bitstream. Bitwise XOR operations are usually involved used in these functions, which can result in outputs of zero. In DNA-XOR, DNA symbols A, T, C, and G are XOR-ed based on the rules in Table 2. In addition to XOR-ing two bits at once which increases hashing speed, it also negates the problem of having zero-valued outputs. However, this comes at a slight computational overhead when converting an input message to a DNA sequence.

By using designing the state transition rules for the DCFSA model based on DNA sequences, security issues of the logistic map such as a small chaotic range and low complexity are addressed. The DNA sequence also helps to overcome security issues in conventional sponge and chaos-based hash functions, which includes weak diffusion, low throughput (specifically for chaos-based hash functions) and hash length flexibility.

The DCFSA map was configured to optimize chaotic performance with respect to DNA computing as well as the sponge construction. Four machine states and four chaotification methods were selected to maximize randomness. The DCFSA map is responsible for generating new DNA sequences to update the sponge state in each round. Unlike other chaotic hash functions which uses the input message to perturb the chaotic points and control parameters, the proposed hash function uses the input message to control the transitions between the machine states. This results in high sensitivity, randomness and near-ideal statistical properties. However, as DCFSA is implemented using floating point number rather than integers or fixed point, it still has room for improvement in terms of efficiency.

The sponge construction is a provably secure cryptographic primitive as discussed in Section II-A. With proper selection of the capacity size, it is indifferentiable from a random oracle and other sponge functions [55]. The proposed hash function has the advantage of extending this provable security notion when producing hash values of up to 128 bits (due to a capacity size of 256 bits). For larger hash values such as 256 or 384 bits, this notion no longer holds. Fortunately, the proposed hash function can be easily modified to accommodate larger capacity sizes to fulfil this requirement if need.

VII. CONCLUSION

In this paper, a new hash function based on three components are proposed: the sponge constriction, DNA computing and chaos. The sponge construction is used to increase the security of the proposed hash function, allowing it to extend provable security notions. DNA computing is used in a new chaotic map to increase its randomness, and also used in the hash function to reduce the number of message blocks

(resulting in improved efficiency) in addition to attaining improved security. The new chaotic map was designed based on a deterministic chaotic finite state automata configuration with four machine states and the logistic map, depicting highly complex chaotic behaviors. The proposed hash function was analysed based on various aspects such as hash value distribution, sensitivity to small message changes, confusion and diffusion properties, robustness against birthday attacks, keyspace analysis, collision resistance, efficiency, and flexibility. Results show that the proposed function has near-perfect statistical properties as compared to other chaotic hash functions.

CONFLICT OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

[1] M. Alawida, J. S. Teh, A. Samsudin, and W. H. Alshoura, "An image encryption scheme based on hybridizing digital chaos and finite state machine," *Signal Process.*, vol. 164, pp. 249–266, Nov. 2019.

[2] S. M. Ismail, L. A. Said, A. G. Radwan, A. H. Madian, and M. F. Abu-ElYazeed, "A novel image encryption system merging fractional-order edge detection and generalized chaotic maps," *Signal Process.*, vol. 167, Feb. 2020, Art. no. 107280.

[3] S. S. Jamal, A. Anees, M. Ahmad, M. F. Khan, and I. Hussain, "Construction of cryptographic S-boxes based on mobius transformation and chaotic tent-sine system," *IEEE Access*, vol. 7, pp. 173273–173285, 2019.

[4] A. Razaq, H. Alolaiyan, M. Ahmad, M. A. Yousaf, U. Shuaib, W. Aslam, and M. Alawida, "A novel method for generation of strong substitution-boxes based on coset graphs and symmetric groups," *IEEE Access*, vol. 8, pp. 75473–75490, 2020.

[5] W. H. Alshoura, Z. Zainol, J. S. Teh, and M. Alawida, "A new chaotic image watermarking scheme based on SVD and IWT," *IEEE Access*, vol. 8, pp. 43391–43406, 2020.

[6] M. Alawida, A. Samsudin, and J. S. Teh, "Enhanced digital chaotic maps based on bit reversal with applications in random bit generators," *Inf. Sci.*, vol. 512, pp. 1155–1169, Feb. 2020.

[7] B. Liu, H. Xiang, and L. Liu, "Reducing the dynamical degradation of digital chaotic maps with time-delay linear feedback and parameter perturbation," *Math. Problems Eng.*, vol. 2020, Feb. 2020, Art. no. 4926937.

[8] A. Jabbari and J. B. Mohasefi, "Improvement in new three-party-authenticated key agreement scheme based on chaotic maps without password table," *Nonlinear Dyn.*, vol. 95, no. 4, pp. 3177–3191, Mar. 2019.

[9] D. Abbasinezhad-Mood and M. Nikooghadam, "Efficient anonymous password-authenticated key exchange protocol to read isolated smart meters by utilization of extended chebyshev chaotic maps," *IEEE Trans. Ind. Informat.*, vol. 14, no. 11, pp. 4815–4828, Nov. 2018.

[10] M. Ahmad, S. Singh, and S. Khurana, "Cryptographic one-way hash function generation using twelve-terms 4D nonlinear system," *Int. J. Inf. Technol.*, vol. 17, pp. 1–9, May 2018.

[11] M. Alawida, J. S. Teh, D. P. Oyinloye, W. H. Alshoura, M. Ahmad, and R. S. Alkhawaldeh, "A new hash function based on chaotic maps and deterministic finite state automata," *IEEE Access*, vol. 8, pp. 113163–113174, 2020.

[12] J. S. Teh, M. Alawida, and J. J. Ho, "Unkeyed hash function based on chaotic sponge construction and fixed-point arithmetic," *Nonlinear Dyn.*, vol. 1, pp. 1–17, Feb. 2020.

[13] M. Alawida, A. Samsudin, J. S. Teh, and W. H. Alshoura, "Deterministic chaotic finite-state automata," *Nonlinear Dyn.*, vol. 98, pp. 1–19, Nov. 2019.

[14] Z. Hua and Y. Zhou, "Exponential chaotic model for generating robust chaos," *IEEE Trans. Syst., Man, Cybern. Syst.*, early access, Aug. 28, 2019, doi: 10.1109/TSMC.2019.2932616.

[15] J. Zheng, H. Hu, and X. Xia, "Applications of symbolic dynamics in counteracting the dynamical degradation of digital chaos," *Nonlinear Dyn.*, vol. 94, no. 2, pp. 1535–1546, Oct. 2018.

[16] M. Alawida, A. Samsudin, and J. S. Teh, "Enhancing unimodal digital chaotic maps through hybridisation," *Nonlinear Dyn.*, vol. 96, no. 1, pp. 601–613, Apr. 2019.

[17] M. Alawida, A. Samsudin, J. S. Teh, and W. H. Alshoura, "Digital cosine chaotic map for cryptographic applications," *IEEE Access*, vol. 7, pp. 150609–150622, 2019.

[18] L. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, vol. 266, no. 5187, pp. 1021–1024, Nov. 1994.

[19] G. Xiao, M. Lu, L. Qin, and X. Lai, "New field of cryptography: DNA cryptography," *Sci. Bull.*, vol. 51, no. 12, pp. 1413–1420, Jun. 2006.

[20] W. Feng, Y. He, H. Li, and C. Li, "A plain-image-related chaotic image encryption algorithm based on DNA sequence operation and discrete logarithm," *IEEE Access*, vol. 7, pp. 181589–181609, 2019.

[21] X. Chai, Z. Gan, K. Yang, Y. Chen, and X. Liu, "An image encryption algorithm based on the memristive hyperchaotic system, cellular automata and DNA sequence operations," *Signal Process., Image Commun.*, vol. 52, pp. 6–19, Mar. 2017.

[22] J. Wu, X. Liao, and B. Yang, "Image encryption using 2D Hénon-sine map and DNA approach," *Signal Process.*, vol. 153, pp. 11–23, Dec. 2018.

[23] X. Chai, Y. Chen, and L. Broyde, "A novel chaos-based image encryption algorithm using DNA sequence operations," *Opt. Lasers Eng.*, vol. 88, pp. 197–213, Jan. 2017.

[24] Z. Liu, C. Wu, J. Wang, and Y. Hu, "A color image encryption using dynamic DNA and 4-D memristive hyper-chaos," *IEEE Access*, vol. 7, pp. 78367–78378, 2019.

[25] N. Iqbal, S. Abbas, M. A. Khan, T. Alyas, A. Fatima, and A. Ahmad, "An RGB image cipher using chaotic systems, 15-puzzle problem and DNA computing," *IEEE Access*, vol. 7, pp. 174051–174071, 2019.

[26] C. S. Chum and X. Zhang, "Hash function-based secret sharing scheme designs," *Secur. Commun. Netw.*, vol. 6, no. 5, pp. 584–592, May 2013.

[27] L. P. Perin, G. Zambonin, D. M. B. Martins, R. Custodio, and J. E. Martina, "Tuning the winternitz hash-based digital signature scheme," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2018, pp. 537–542.

[28] W. Luo, Y. Hu, H. Jiang, and J. Wang, "Authentication by encrypted negative password," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 1, pp. 114–128, Jan. 2019.

[29] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 6th ed. Upper Saddle River, NJ, USA: ACM Press, 2013.

[30] T. Fox-Brewster. (2017). *Google Just 'Shattered' An Old Crypto Algorithm-Here's Why That's Big For Web Security*. [Online]. Available: https://www.theverge.com/2017/2/23/14712118/google-sha1-collision-broken-we%b-encryption-shatteredurldate2019-11-27

[31] S. Deng, Y. Li, and D. Xiao, "Analysis and improvement of a chaos-based hash function construction," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 15, no. 5, pp. 1338–1347, May 2010.

[32] A. Sotirov, M. Stevens, J. Appelbaum, A. K. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger, "MD5 considered harmful today, creating a rogue CA certificate," in *Proc. 25th Annu. Chaos Commun. Congr.*, 2008, p. EPFL-CONF-164547. [Online]. Available: http://www.win.tue.nl/hashclash/rogue-ca/

[33] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu, "Cryptanalysis of the hash functions md4 and ripemd," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 2005, pp. 1–18.

[34] M. J. Dworkin, "SHA-3 standard: Permutation-based hash and extendable-output functions," Federal Inf. Process. Standards, Tech. Rep. (NIST FIPS)-US-202, Aug. 2015.

[35] J. Guo, G. Liao, G. Liu, M. Liu, K. Qiao, and L. Song, "Practical collision attacks against round-reduced SHA-3," *J. Cryptol.*, vol. 33, pp. 228–270, Jan. 2019.

[36] D. Saha, S. Kuila, and D. R. Chowdhury, "Symsum: Symmetric-sum distinguishers against round reduced SHA3," in *Proc. IACR*, 2017, pp. 240–258.

[37] S. Huang, X. Wang, G. Xu, M. Wang, and J. Zhao, "New distinguisher on reduced-round keccak sponge function," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. E102.A, no. 1, pp. 242–250, 2019.

[38] M. Amy, O. Di Matteo, V. Gheorghiu, M. Mosca, A. Parent, and J. Schanck, "Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3," in *Selected Areas in Cryptography—SAC* (Lecture Notes in Computer Science), vol. 10532, R. Avanzi and H. Heys, Eds. Cham, Switzerland: Springer, 2016, pp. 317–337.

[39] Y. Li, D. Xiao, and S. Deng, "Secure hash function based on chaotic tent map with changeable parameter," *High Technol Lett*, vol. 18, no. 1, pp. 7–12, 2012.

[40] J. Liu, X. Wang, K. Yang, and C. Zhao, "A fast new cryptographic hash function based on integer tent mapping system," *J. Comput.*, vol. 7, no. 7, pp. 1671–1680, Jul. 2012.

[41] J. S. Teh, K. Tan, and M. Alawida, "A chaos-based keyed hash function based on fixed point representation," *Cluster Comput.*, vol. 22, no. 2, pp. 649–660, Jun. 2019.

[42] A. Akhshani, S. Behnia, A. Akhavan, M. A. Jafarizadeh, H. Abu Hassan, and Z. Hassan, "Hash function based on hierarchy of 2D piecewise nonlinear chaotic maps," *Chaos, Solitons Fractals*, vol. 42, no. 4, pp. 2405–2412, Nov. 2009.

[43] A. Akhavan, A. Samsudin, and A. Akhshani, "A novel parallel hash function based on 3D chaotic map," *EURASIP J. Adv. Signal Process.*, vol. 2013, no. 1, p. 126, Dec. 2013.

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

[45] Y. Li and X. Li, "Chaotic hash function based on circular shifts with variable parameters," *Chaos, Solitons Fractals*, vol. 91, pp. 639–648, Oct. 2016.

[46] N. Abdoun, S. El Assad, O. Déforges, R. Assaf, and M. Khalil, "Design and security analysis of two robust keyed hash functions based on chaotic neural networks," *J. Ambient Intell. Hum. Comput.*, vol. 15, pp. 1–25, Feb. 2019.

[47] Y. Li, G. Ge, and D. Xia, "Chaotic hash function based on the dynamic S-Box with variable parameters," *Nonlinear Dyn.*, vol. 84, no. 4, pp. 2387–2402, Jun. 2016.

[48] Y. Li, X. Li, and X. Liu, "A fast and efficient hash function based on generalized chaotic mapping with variable parameters," *Neural Comput. Appl.*, vol. 28, no. 6, pp. 1405–1415, Jun. 2017.

[49] H. Liu, A. Kadir, and J. Liu, "Keyed hash function using hyper chaotic system with time-varying parameters perturbation," *IEEE Access*, vol. 7, pp. 37211–37219, 2019.

[50] Z. Lin, S. Yu, and J. Lü, "A novel approach for constructing one-way hash function based on a message block controlled 8D hyperchaotic map," *Int. J. Bifurcation Chaos*, vol. 27, no. 07, Jun. 2017, Art. no. 1750106.

[51] Q. Jiang, L. Wang, and X. Hei, "Parameter identification of chaotic systems using artificial raindrop algorithm," *J. Comput. Sci.*, vol. 8, pp. 20–31, May 2015.

[52] Y. Li, "Collision analysis and improvement of a hash function based on chaotic tent map," *Optik*, vol. 127, no. 10, pp. 4484–4489, May 2016.

[53] M. Ahmad, S. Khurana, S. Singh, and H. D. AlSharari, "A simple secure hash function scheme using multiple chaotic maps," *3D Res.*, vol. 8, no. 2, p. 13, Jun. 2017.

[54] J. S. Teh, M. Alawida, and Y. C. Sii, "Implementation and practical problems of chaos-based cryptography revisited," *J. Inf. Secur. Appl.*, vol. 50, Feb. 2020, Art. no. 102421.

[55] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "On the indifferentiability of the sponge construction," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 2008, pp. 181–197.

[56] J. S. Teh, A. Samsudin, and A. Akhavan, "Parallel chaotic hash function based on the shuffle-exchange network," *Nonlinear Dyn.*, vol. 81, no. 3, pp. 1067–1079, Aug. 2015.

[57] Y. Li and G. Ge, "Cryptographic and parallel hash function based on cross coupled map lattices suitable for multimedia communication security," *Multimedia Tools Appl.*, vol. 78, no. 13, pp. 17973–17994, 2019.

[58] M. Asgari Chenaghlu, S. Jamali, and N. Nikzad Khasmakhi, "A novel keyed parallel hashing scheme based on a new chaotic system," *Chaos, Solitons Fractals*, vol. 87, pp. 216–225, Jun. 2016.

[59] W. Chankasame and W. San-Um, "A chaos-based keyed hash function for secure protocol and messege authentication in mobile ad hoc wireless networks," in *Proc. Sci. Inf. Conf. (SAI)*, Jul. 2015, pp. 1357–1364.

[60] Y. Li, D. Xiao, H. Li, and S. Deng, "Parallel chaotic hash function construction based on cellular neural network," *Neural Comput. Appl.*, vol. 21, no. 7, pp. 1563–1573, Oct. 2012.

[61] Y.-L. Luo and M.-H. Du, "One-way hash function construction based on the spatiotemporal chaotic system," *Chin. Phys. B*, vol. 21, no. 6, Jun. 2012, Art. no. 060503.

[62] M. Todorova, B. Stoyanov, K. Szczypiorski, W. Graniszewski, and K. Kordov, "BentSign: Keyed hash algorithm based on bent Boolean function and chaotic attractor," *Bull. Polish Acad. Sciences. Tech. Sci.*, vol. 67, no. 3, pp. 557–569, 2019.

[63] R. Rivest, "The MD5 message-digest algorithm," RFC Editor, USA, Tech. Rep., 1992.

[64] Y. Wang, K.-W. Wong, and D. Xiao, "Parallel hash function construction based on coupled map lattices," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 16, no. 7, pp. 2810–2821, Jul. 2011.

[65] A. Kanso and M. Ghebleh, "A structure-based chaotic hashing scheme," *Nonlinear Dyn.*, vol. 81, nos. 1–2, pp. 27–40, Jul. 2015.

[66] Z. Lin, C. Guyeux, S. Yu, Q. Wang, and S. Cai, "On the use of chaotic iterations to design keyed hash function," *Cluster Comput.*, vol. 19, pp. 1–15, Oct. 2017.

[67] M. Todorova, B. Stoyanov, K. Szczypiorski, and K. Kordov, "SHAH: Hash function based on irregularly decimated chaotic map," 2018, *arXiv:1808.01956*. [Online]. Available: http://arxiv.org/abs/1808.01956

[68] Y. Wang, X. Liao, D. Xiao, and K.-W. Wong, "One-way hash function construction based on 2D coupled map lattices," *Inf. Sci.*, vol. 178, no. 5, pp. 1391–1406, Mar. 2008.

MOATSUM ALAWIDA received the B.Sc. degree from Mutah University Jordan, in 2005, the M.Sc. degree in information systems from the University of Jordan, in 2010, and the Ph.D. degree in computer science/cybersecurity (cryptography) from the School of Computer Sciences, University Sains Malaysia, in 2020. His research interests include chaotic systems, chaos-based applications, multimedia security, blockchain, cybersecurity, quantum-based cryptography, and cryptography.

AZMAN SAMSUDIN received the B.Sc. degree in computer science from the University of Rochester, NY, USA, in 1989, and the M.Sc. and Ph.D. degrees in computer science from the University of Denver, CO, USA, in 1993 and 1998, respectively. He is currently a Professor with the School of Computer Science, Universiti Sains Malaysia (USM). He has published more than 100 articles over a series of books, professional journals, and conferences. His research interests include cryptography, switching networks, and parallel computing.

NANCY ALAJARMEH received the Ph.D. degree from New Mexico State University, USA, in 2014. She is currently an Assistant Professor of computer science. Her teaching experience covers teaching a number of lower division and upper division classes in computer science. During her work experience at Tafila Technical University (TTU), she has been intensively engaged with various administrative duties and roles, such as heading the Department of Computer and Information Technology for the term 2017–2018 and directing the Office of Human Resources Development and e-Learning at TTU starting from early 2020. Her research interests include human–computer interaction, usable security, educational technology, and data science.

JE SEN TEH received the B.Eng. degree (Hons.) in electronics from Multimedia University, Malaysia, in 2011, the M.Sc. degree in computer science and the Ph.D. degree from Universiti Sains Malaysia, in 2013 and 2017, respectively. He is currently working as a Senior Lecturer with Universiti Sains Malaysia. His research interests include cryptography, cryptanalysis, random number generation, machine learning, and chaos theory.

MUSHEER AHMAD received the B.Tech. and M.Tech. degrees from the Department of Computer Engineering, Aligarh Muslim University, India, in 2004 and 2008, respectively, and the Ph.D. degree in chaos-based cryptography from the Department of Computer Engineering, Jamia Millia Islamia, New Delhi, India. From 2007 to 2010, he has worked with the Department of Computer Engineering, Aligarh Muslim University. He has been working as an Assistant Professor with the Department of Computer Engineering, Jamia Millia Islamia, since 2011. He has published over 85 research articles in international reputed refereed journals and conference proceedings of the IEEE/Springer/Elsevier. He has more than 1200 citations of his research works with an H-index of 20. His research interests include multimedia security, chaos-based cryptography, cryptanalysis, machine learning for security, image processing, and optimization techniques. He has served as a reviewer and a technical program committee member of many international conferences. He has also served as referee of some renowned ISI journals.

WAFA' HAMDAN ALSHOURA received the B.Sc. degree and the M.Sc. degree in computer science from Al-Zaytoonah University, Jordan, in 2012 and 2017, respectively. She is currently pursuing the Ph.D. degree with the School of Computer Sciences, University Sains Malaysia. Her research interests include digital watermarking and hash function.