Received May 22, 2020, accepted June 10, 2020, date of publication June 16, 2020, date of current version June 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3002763

A New Hash Function Based on Chaotic Maps and Deterministic Finite State Automata

MOATSUM ALAWIDA[1], JE SEN TEH[1], DAMILARE PETER OYINLOYE[1], WAFA' HAMDAN ALSHOURA[1], MUSHEER AHMAD[2], AND RAMI S. ALKHAWALDEH[3]

[1]School of Computer Sciences, Universiti Sains Malaysia (USM), Gelugor 11800, Malaysia
[2]Department of Computer Engineering, Jamia Millia Islamia, New Delhi 110025, India
[3]Department of Computer Information Systems, The University of Jordan, Aqaba 77110, Jordan

Corresponding authors: Moatsum Alawida (matsm88@yahoo.com) and Je Sen Teh (jesen_teh@usm.my)

This work was supported in part by the Ministry of Education Malaysia through the Fundamental Research Grant Scheme (FRGS) under Project FRGS/1/2019/ICT05/USM/02/1.

ABSTRACT In this paper, a new chaos-based hash function is proposed based on a recently proposed structure known as the deterministic chaotic finite state automata (DCFSA). Out of its various configurations, we select the forward and parameter permutation variant, $DCFSA_{FWP}$ due to its desirable chaotic properties. These properties are analogous to hash function requirements such as diffusion, confusion and collision resistance. The proposed hash function consists of six machine states and three simple chaotic maps. This particular structure of DCFSA can process larger message blocks (leading to higher hashing rates) and optimizes its randomness. The proposed hash function is analyzed in terms of various security aspects and compared with other recently proposed chaos-based hash functions to demonstrate its efficiency and reliability. Results indicate that the proposed hash function has desirable statistical characteristics, elevated randomness, optimal diffusion and confusion properties as well as flexibility.

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

I. INTRODUCTION

A hash function is a one-way function that compresses messages to fixed-length hash values. Hash functions are widely used in various cryptographic applications [1]–[7]. Blockchain and cryptocurrency are also heavily dependent on hash functions in their consensus protocols [8]. The Merkle-Damgard (MD) structure is one of the most well-known hash constructions and has been used to design hash functions such as MD5 and SHA-1. However, some conventional hash functions have shown security defects under differential or collision attacks [9], [10]. Furthermore, a collision has been found for SHA-1 [11]. With the cryptanalysis of MD5 and a theoretical break of SHA-1 [12], the National Institute of Standards and Technology (NIST) released the SHA-2 family. However, SHA-2's design is based on similar principles of MD5 and SHA-1, and the same class of attacks has used to cryptanalyze SHA-2 [13].

NIST called for a competition to establish a new, standardized hash, henceforth known as SHA-3 [14]. A hash function based on the sponge structure was finally selected [15]. Since then, the SHA-3 algorithm has been subjected to various cryptanalytic attacks [16]–[18]. Although SHA-3 has yet to be fully cryptanalyzed, alternative hash function designs should still be explored in the meantime [19].

Chaotic systems have been used to design hash functions due to the commonalities between chaotic systems and hash function characteristics [20]. These characteristics include sensitivity to small changes to initial conditions and system parameters, random-like behavior, ergodicity, diffusion and confusion properties. Thus, various hash function constructs have been proposed based on the chaotic systems such as iterating simple chaotic maps [21], [22], multiple maps [23], high-dimensional maps [24], [25] and message block-controlled hyperchaotic map [26]. Other chaos-based hash functions in literature include Li *et al.*'s hash function based on generalized chaotic maps with perturbed chaotic parameters [27], Ahmad *et al.* design based on the dynamics of a nonlinear 12-term 4D chaotic system [28], Liu *et al.*'s proposal based on the Lorenz system with multiple parameters and time-varying perturbation [29].

Akhavan *et al.* have proposed a chaotic hash function based on piecewise nonlinear chaotic map and a 1D chaotic map in parallel mode [24] whereas Teh *et al.* have proposed a keyed

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

hash function based on logistic map realized with fixed point representation [30]. Li *et al.* have proposed a new chaotic hash function based on circular shifts and variable parameters, piecewise linear chaotic map and one-way coupled map lattice [31]. The maps were used generate chaotic variables while circular shifts were used in enhancing randomness. However, most of these proposed functions suffer from security flaws due to weaknesses in the underlying chaotic maps or designs used. These security flaws can be generalized as follow:

- Using 1D chaotic map without enhancing chaotic behaviours lead security defects and ease of estimating chaotic parameters [32]–[36].
- Using hyperchaotic maps or HD chaotic maps lead to high computational complexity [23], [24], [26], [28].
- Perturbation of initial conditions, or chaotic points, or chaotic parameters in each chaotic iteration and message block lead more overheads [24], [29], [31].
- A taken message block in each iteration is limited by floating point precision [37], [38].
- Convoluted or complex designs lead to a lack of analyzability or hidden weaknesses [37].

In this paper, we propose a new chaotic hash function based on deterministic chaotic finite state automata (DCFSA) with dynamic perturbation. DCFSA was originally proposed to enhance 1D chaotic behaviors without using external entropy sources [39], [40]. Out of its various configurations, we select $DCFSA_{FWP}$ due to its various advantages such as large chaotic parameter range, high complexity, better randomness, and elevated nonlinearity. Six machine states and two transitions between each state are used to reduce number of chaotic iterations and message block iterations. $DCFSA_{FWP}$ is iterated using the perturbed buffer values to generate new chaotic points. The hash value is then extracted from the binary values of these chaotic points. The hash function has the flexibility to generate hash values of different lengths, all of which are uniformly distributed. We then compare the proposed hash function with other existing chaotic hash functions, whereby findings show that the proposed function has desirable statistical properties, is collision resistant, efficient and has a simple analyzable design to facilitate future cryptanalysis efforts.

The remaining sections of this paper are as follows: Section II provides details about DCFSA architectures and its chaotic analyses. Then, the proposed chaos-based hash function is detailed in Section III, followed by its security and performance analysis in Section IV. Some final remarks and a summary of findings conclude the paper in Section VII.

II. DETERMINISTIC CHAOTIC FINITE STATE AUTOMATA

DCFSA combines 1D chaotic maps with deterministic finite automata (DFA) to overcome dynamical degradation and enhance chaotic complexity [39]. 1D chaotic maps such as the logistic map, sine map, and tent map are known as unimodal maps since they only have one critical point for their control parameter. Due to this property, using them directly in cryptographic applications without any enhancements can lead to security problems [34], [41], [42]. The main advantage using DCFSA as a chaotification strategy is that the computational complexity of the resulting chaotic system is still comparable to a unimodal map, whereby each iteration of the DCFSA only involves one chaotic map without additional calculations. Therefore, DCFSA has better chaotic performance as compared to unimodal maps while maintaining a low computational complexity.

DFA includes a number of machine states and transition arrows between states. Each transition can hold a single symbol and a state transition rule can be used to design various DFA configurations. DCFSA also depends on similar factors that dictate its configuration which include the number of machine states and state transition rule but includes additional ones such as the number of chaotic maps, symbols and the type of perturbation operation that is associated with each state transition. Out of the ten DCFSA configurations analyzed in [39], we have selected $DCFSA_{FWP}$ to be used in the proposed hash function as it depicts optimal chaotic characteristics. The acronym *FWP* refers to the type of perturbations that are associated with each state transition, which are forward perturbation (*FW*) and parameter perturbation (*P*). Both of these operations are detailed in the following subsection.

A. $DCFSA_{FWP}$ CONFIGURATION

The $DCFSA_{FWP}$ used in this work will utilize three 1D chaotic maps, logistic, sine, and tent, that are mathematically represented as

$$x_{n+1} = r_1 x_n (1 - x_n) \tag{1}$$

$$x_{n+1} = \frac{r_2}{4} sin(\pi x_n) \tag{2}$$

$$x_{n+1} = \begin{cases} \frac{r_3}{2} x_n & \text{if } x_n < 0.5 \\ \frac{r_3}{2}(1 - x_n) & \text{if } x_n \geq 0.5 \end{cases} \tag{3}$$

respectively, where x_0 is the initial condition, $\{r_1, r_2, r_3\} \in [0, 4]$ are the control parameters, n is the number of iterations, and $x_n \in [0, 1]$ denotes the system variable or chaotic point.

The standard state transition rule, δ dictates that machine states are linked to one another by two transitions, labeled as 0 and 1 in Figure 1 (note that the initial state is indicated using the dashed line). Three machine states and six transition arrows are used to generate the standard DFA. δ can be defined as

- $q_1 \times 0 \rightarrow q_2$
- $q_1 \times 1 \rightarrow q_3$
- $q_2 \times 0 \rightarrow q_3$
- $q_2 \times 1 \rightarrow q_1$
- $q_3 \times 0 \rightarrow q_1$
- $q_3 \times 1 \rightarrow q_2$

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

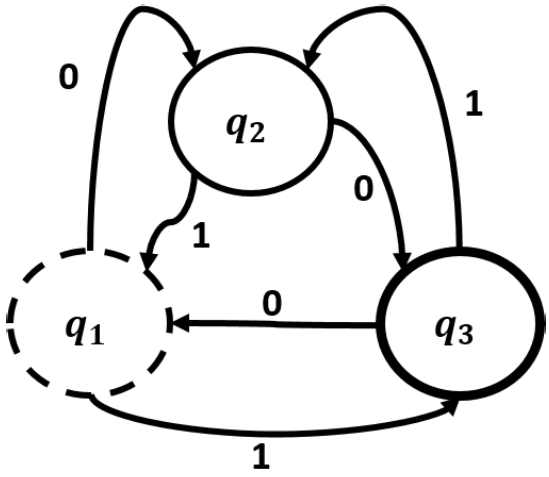

FIGURE 1. DFA with three states.

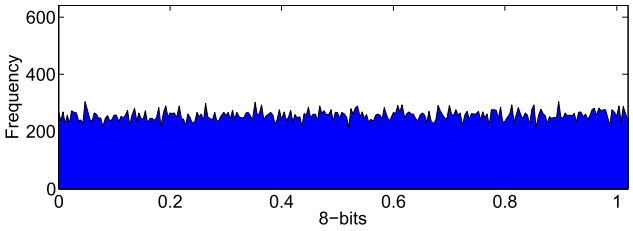

FIGURE 2. Frequency of phase space of logistic map when $r = 4$.

$DCFSA_{FWP}$ involves forward and parameter perturbations and three different 1D chaotic maps. Each machine state is associated with one chaotic map and each state has an internal buffer to store the value of the chaotic point after each iteration. During the transition between states, a quantization function G selects the type of perturbation to be performed. This selection is based on the value of the chaotic point after each iteration of the chaotic map. G is denoted as

$$G = \begin{cases} 1 & if\ x_n > T \\ 0 & if\ x_n \leq T \end{cases} \tag{4}$$

where x_n is a chaotic point, T is the threshold value set as 0.5 to divide the phase space equally. The three 1D chaotic maps have a control parameter of $r = 3.999$ to ensure they have uniform distribution and ergodicity as depicted in Figure 2. We can see that the phase space is divided to 256 intervals (8-bit) and logistic is iterated to 10,000 times. The intervals are equally visited, that means the system is ergodic and uniform distribution. $DCFSA_{FWP}$ can generate chaotic behaviors along $r \in (0, 4]$ and is ergodic [39]. Thus, we set $T = 0.5$ to equal selection between two type of perturbation methods. That will help to make balance between two types and generate better chaotic performance. Forward and parameter perturbation will be selected if $G = 1$ and $G = 0$, respectively. Figure 3 shows the proposed $DCFSA_{FWP}$ configuration along with its corresponding buffer values.

According to the DCFSA description in [39], forward perturbation is used to modify chaotic points after iterating

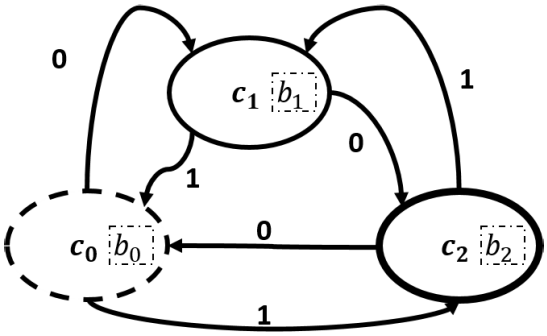

FIGURE 3. Buffers in DCFSA for perturbation purposes.

chaotic maps in each machine state. The chaotic point from the prior machine state is used in perturbation function. The forward perturbation function is defined as

$$x_{n+1} = F_{fw}(c_i, r_i, x_n, b_i) = (c_i(r_i, b_i) + x_n)\ mod\ 1 \tag{5}$$

where i is the index of the current machine state, b_i is the value stored in the buffer (which corresponds to the last chaotic point generated by the current machine state), and x_n is the chaotic point generated by the prior machine state. $c_i(r_i, b_i)$ represents the chaotic map of the current machine state, whose control parameter and chaotic points are r_i and b_i, respectively.

In parameter perturbation, the previous chaotic point, x_n (from a prior machine state $i-1$) is used to perturb the control parameter of the chaotic map in the current state, i. Parameter perturbation can be defined as

$$x_{n+1} = F_{pr}(c_i, r_i, x_n, b_i) = c_i(\phi(r_i, x_n), b_i) \tag{6}$$

where the perturbation function, ϕ is the parameter scaling function whose outputs are limited to values that have positive Lyapunov exponents (LE). ϕ is calculated as

$$\phi(r_i, x_n) = ((r_i + (1 - x_n)\zeta))\ mod\ (\zeta) + b_{min} \tag{7}$$

where b_{max} and b_{min} are the chaotic range of c_i, where $\zeta = b_{max} - b_{min}$. $DCFSA_{FWP}$ is a combination of two robust chaotification methods, leading to improved chaotic properties. The perturbation methods and state buffer values are used to generate new chaotic points. The selection between the two perturbation methods is based on the previous chaotic point (from the previous state) and the threshold value. Therefore, $DCFSA_{FWP}$ is a piecewise function and can be written as

$$x_{n+1} = \begin{cases} (c_i(r_i, b_i) + x_n)\ mod\ 1 & if\ \ x_n > T \\ c_i(\phi(r_i, x_n), b_i) & if\ \ x_n \leq T \end{cases} \tag{8}$$

A slight change to one bit of any chaotic point leads to a strong avalanche effect that propagates from one machine state to another. More details about DCFSA can be obtained from [39].

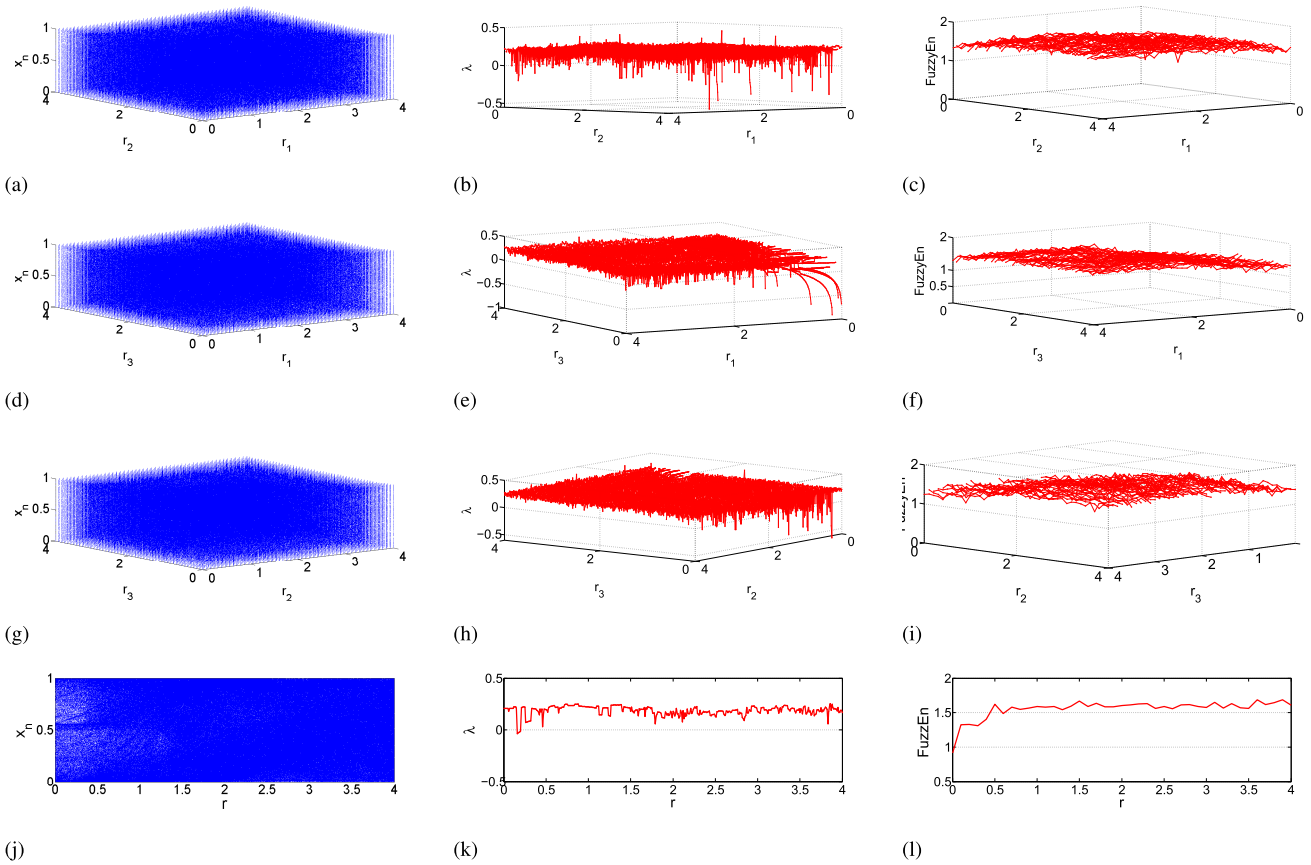

FIGURE 4. Bifurcation and LE, and FuzzyEn diagrams of $DCFSA_{FWP}$ with different control parameters, (a) Bifurcation diagram r_1 & r_2, (b) LE values r_1 & r_2, (c) FuzzEn values r_1 & r_2, (d) Bifurcation diagram r_1 & r_3, (e) LE values r_1 & r_3, (f) FuzzEn values r_1 & r_3, (g) Bifurcation diagram r_2 & r_3, (h) LE values r_2 & r_3, (i) FuzzEn values r_2 & r_3, (j) Bifurcation diagram when $r_1 = r_2 = r_3$, (k) LE values when $r_1 = r_2 = r_3$, (l) FuzzEn values when $r_1 = r_2 = r_3$.

B. $DCFSA_{FWP}$ ANALYSIS

$DCFSA_{FWP}$ depicts highly chaotic behavior and high sensitivity to small changes to its parameters, along with a wide range of chaotic parameters. To depict these chaotic properties, we use bifurcation diagrams, LE and fuzzy entropy (FuzzyEn). The bifurcation diagram is a classical indicator of chaotic range whereas LE is used to evaluate the system's chaotic sensitivity. Large positive values of LE indicate high sensitivity and chaotic behavior. FuzzyEn is used as a measure of complexity, which is a metric that can be used to determine if a chaotic system is susceptible to estimation parameter attacks. A high FuzzyEn value is proportionate to the complexity of the system.

Figure 4 shows the bifurcation, LE and FuzzyEn diagrams for different parameter settings of the three underlying 1D chaotic maps and same control parameters. The first three rows of Subfigures 4 (a) to (f) show chaotic behaviors when two control parameters are changed while one remains constant. This allows us to study the relationship between the various chaotic parameters. We can see that $DCFSA_{FWP}$ has a large chaotic parameter range within the standard range of $(0, 4]$. Thus, there is higher flexibility when it comes to selecting suitable control parameters that guarantee chaotic behavior. LE diagrams demonstrate that $DCFSA_{FWP}$ has positive values for different parameter settings, which implies that it is a highly sensitive system. Some outlying negative values can be observed in 4 (b) and (e) because two control parameters are dynamically changed while one remains constant in these experiments. When all three parameters are dynamically changed, the system depicts positive LE values. The FuzzyEn diagrams show that $DCFSA_{FWP}$ has highly complex patterns and irregular behavior. Subfigures 4 (j), (k) and (l) depict $DCFSA_{FWP}$ behaviors when the same control parameter value is applied to all three chaotic maps. We can see that $DCFSA_{FWP}$ still produces chaotic behaviors along the range of $(0, 4]$, thus confirming that $DCFSA_{FWP}$ based on two chaotification methods can enhance the properties of 1D chaotic maps.

III. DCFSA-BASED HASH FUNCTION

The DCFSA chaotification method enhances the chaotic behaviour of classical maps, leading to chaotic trajectories with long cycle lengths [40]. Due to these desirable properties, DCFSA can be adopted to design new cryptographic hash function with high security. In the proposed work, we employ a $DCFSA_{FWP}$ configuration with six machine states, $\{q_0, q_1, .., q_4, q_5\}$, three chaotic maps, $\{c_0, c_1, c_2\}$ and an updated state transition mapping as shown in Figure 5.

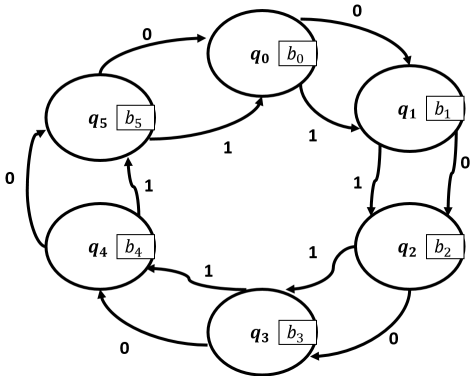

FIGURE 5. $DCFSA_{FWP}$ with 6 machine states (logistic map - $\{q_0,q_3\}$, tent map - $\{q_1,q_4\}$, sine map - $\{q_2,q_5\}$).

The quantization function is based on a threshold value $T = 0.5$, while 1 and 0 denote forward and parameter perturbation respectively. The states q_0 and q_3 utilize the logistic map c_0, q_1 and q_4 utilize the tent map c_1, and q_2 and q_5 utilize the sine map c_2. Each map will be assigned different control parameters and initial buffer values which can be any value within its chaotic range. The steps of the overall hash function algorithm based on $DCFSA_{FWP}$ can be described as follows:

Algorithm 1 Padding Algorithm

Data: Input message Mes_{bit}.
Result: Message blocks Mes_{bit}, Number of blocks M

$N = length(Mes_{bit})$;
$remainder = N \ mod \ 312$;
$LenBin = dec2bin(N, 32)$;
if $remainder! = 0$ **then**
  $Mes_{bit}(N+1) =' 1'$;
  for $i = N+2$ *to* $N + remainder \ : +1$ **do**
    $Mes_{bit}(i) =' 0'$;
$N = N + remainder$;
$Mes_{bit}(N+1 : N+32) = LenBin(1 : 32)$;
$Mes_{bit}(N+33) =' 1'$;
for $i = N+34$ *to* $N+312$ **do**
  $Mes_{bit}(i) =' 0'$;
$M = N/312 + 1$;

1) Transform a message of any length into binary, ($Mes \rightarrow Mes_{bit}$) by using ASCII representation.
2) A conventional padding rule is then used to ensure that the overall message is a multiple of 312 bits. The padding also includes the original message length to avoid collisions and length extension attacks. First, a single 1 bit is appended to the end of the message, followed by 0s. The last 32 bits of the block is reserved for the message length. If a message cannot accommodate the 32 bits for message length or if the message is a multiple of 312 bits, an entirely new block will be created. A detailed look is available in Algorithm 1. Let $M = \frac{N}{312}$ be the number of 312-bit message blocks. Each message block will undergo one *round* of $DCFSA_{FWP}$, which is defined as a transition from q_0 to q_5 then back to q_0 for a total of 6 map iterations.
3) Divide each 312-bit message block $Mes_{bit}(1:312) \rightarrow ch^j$, ch^j into six 52-bit sub-blocks ch_i^j, where $j = 0, .., 5$ and $i = 1, \ldots M$. These 52-bit sub-blocks are then used as $U(0, 52)$ fixed point numbers, where U is an unsigned number with 0 bits to represent integers and 52 bits to represent the fractional portion of a real number. The fixed point representation of each 52-bit fixed point number is calculated as
$$ch_i^j = \sum_{k=1}^{52} (ch_i^j(k) \times 2^{-k}) \tag{9}$$
where ch_i^j are real numbers that range between 0 and 1 to ensure that the DCFSA operates within its phase space. We generate six values, one for each machine state. Thus, the proposed hash function processes 312 bits each round, and each block ch_i^j has an effect on subsequent blocks. We utilize fixed point numbers as it is more efficient to compute as compared to IEEE 754 floating point number, and also prevents other implementation problems commonly associated with floating point numbers [37].
4) By using modular addition, ch_i^j is used to modify each of the buffer values as $b_j = ((ch_i^j + b_j) \times 7^{14}) \ mod \ 1$. The modified buffer values will then be used to calculate future chaotic points if the machine state is visited again. The constant 7^{14} is used to increase diffusion property and amplify the effect of the small changes to ch_i^j. Modular addition ensures that the buffer values will remain as fractional values that range between 0 and 1 to ensure that the map does not iterate out of scope.
5) Iterate $DCFSA_{FWP}$ with b_j six times (one round) to generate new buffer values for each of the machine states. These six iterations will involve each of the individual machine states one by one due to how the state transition rule was designed (i.e. $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q3 \rightarrow q_4 \rightarrow q_5$). Each b_j is modified twice, once when being perturbed by the message bits and second due to the $DCFSA_{FWP}$ iterations. During the $DCFSA_{FWP}$ iterations, the buffer values are perturbed using one of two perturbation techniques which are dynamically selected based on various factors such as message bits, key values, and past iterations. The chaotic properties of $DCFSA_{FWP}$ eliminates statistical correlation between current and past buffer values.
6) Repeat steps 3-5 for each message block. The entire process can be seen as a compression function that compresses all message bits to produce the final set of buffer values.
7) To increase the diffusion effect of the proposed hash function, we iterate $DCFSA_{FWP}$ an additional 50 rounds (6 iterations per round). The 50 rounds increase the differences between all chaotic points,

fully diffusing the effect of all message blocks to all state buffers.

8) In the final step, we generate the hash value, where H denotes its length. $DCFSA_{FWP}$ is iterated $\lceil \frac{H}{32} \rceil$ times, then converts each chaotic point into its binary representation. The 32 most significant bits of each chaotic point is then extracted to be part of the hash. For example, to generate a 128-bit hash value, four iterations of $DCFSA_{FWP}$ are required. This implies that four machine states are visited, their chaotic maps are iterated, and the resulting chaotic points are collected to form the hash value. It can be seen that the proposed hash function can trivially produce hash values of other lengths by varying the number of iterations.

In short, the hash function involves perturbing the buffer values of each machine state with message bits, then iterating the DCFSA such that each state is visited multiple times. The random-like nature of the chaotic points being produced will then randomly select between forward and parameter perturbation, which is an additional dimension of unpredictability. Since the final hash value is derived from the chaotic points being produced by the DCFSA, generating varying lengths of hash values is a straightforward process. All steps involved in the proposed hash function are included in Algorithm 2.

Algorithm 2 DCFSA-Based Hash Function Algorithm

Data: Input message Mes_{bit}. Message blocks Mes_{bit}, Number of blocks M, b_j and $r_j, j = 0, 1, 2, 3, 4, 5$

Result: H hash value

$N = length(Mes_{bit})$;
$M = N/312 + 1$;
Call Padding algorithm;
for $i = 1 to M$ **do**
  $ch_i = Mes_{bit}(312 \times (i-1) + 1 : i \times 312)$;
  for $j = 0 to 5$ **do**
  $ch_i^j = ch_i(52 \times (j) + 1 : (j+1) \times 52)$;
  for $k = 1 to 52$ **do**
    $ch_i^j = (ch_i^j(k) \times 2^{-k})$;
  $b_j = ((ch_i^j + b_j) \times 7^{14}) \; mod \; 1$;
  $b_j = DCFSA_{FWP}(r_j, b_j)$;
for $i = 1$ *to* 50 **do**
  for $j = 0$ *to* 5 **do**
    $b_j = DCFSA_{FWP}(r_j, b_j)$;
for $n = 1$ *to* $\lceil \frac{H}{32} \rceil$ **do**
  $b_n = DCFSA_{FWP}(r_n, b_n)$;
  $H = (b_n)_{1,\ldots,32}$;

IV. EXPERIMENTAL EVALUATION

In this section, the performance of the proposed DCFSA-based hash function is analyzed. The input message for all experiments is a sequence of the letter a. All experiments are implemented using Matlab R2012b on a Intel Core i5-560M @ 2.67GHz Processor, 8GB RAM and Windows 7 operating system. The proposed scheme is evaluated under a standardized set of statistical tests which includes distribution analysis, sensitivity test, diffusion and confusion test and collision analysis. The use of these metrics also allows for a fair comparison against the state-of-the-art in chaos-based hash functions. We use six chaotic maps that have a total of six initial conditions and six control parameters. The hash length for all experiments is 128 bits. The initial buffer values are set to $b_i = 0.25$ whereas the control parameter values are set to $r_i = 3.99$ for $i = \{0, 1, .., 5\}$.

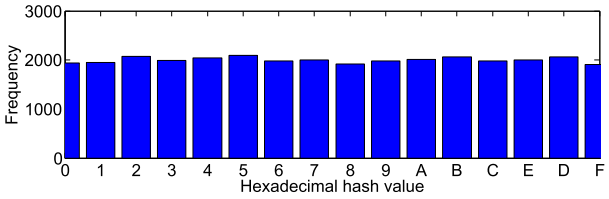

FIGURE 6. Distribution hexadecimal hash value for 1000 different input messages.

A. DISTRIBUTION OF HASH VALUE

A uniformly distributed output is an important requirement for a secure cryptographic hash function. Any biases can be leveraged upon by an adversary to perform collision or forgery attacks. To analyze the distribution of the proposed hash function, 1000 randomly selected input messages of 1500 characters are hashed and their corresponding hash values are represented using hexadecimal values. The frequency of occurrence for each hexadecimal value is noted. The frequency of occurrence for each value is shown in Figure 6, depicting an even distribution.

B. SENSITIVITY TEST

Hash functions should be highly sensitivity to any slight change to its input message, initial values and system parameters. Thus, high sensitivity is a desirable trait of a hash function. In this test, an input message consisting of 1000 'a' characters is selected. Subsequently, a number of slight changes in to the input message, initial values or system parameters are performed and their corresponding hash values are calculated. Hash values for the following cases are computed:

Case 1 : An input message M consisting of 1000 '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 initial value $x_0 + 10^{-15}$
Case 6 : Small change to system parameter $r + 10^{-15}$

Table 1 shows the hexadecimal representation of six hash values under the various cases, indicating a high level of sensitivity to its inputs. The percentage of bits changed for each case compared to case 1 is included into the table,

TABLE 1. 128 bits hash values in different six cases and percentage of number of bits changed.

Case	Hash value	Percentage of number of bits changed
Case 1	9c6e46eb7bf7a33485db8ea55b6eda36	
Case 2	0b26d26f046cc4def7ddbbd27a2edf81	47.65625
Case 3	5a4bb1d741355215bd84d7b84100ce30	47.65625
Case 4	d60d6c385fa13b86ae227bdbbc02dad0	50.78125
Case 5	7e1ba9656d5c15a0b19807c9ca1cc288	50
Case 6	f7958692f2b5fa50ded81f60c6f80540	51.5625

depicting the sensitivity and strong avalanche property of the proposed hash function.

$DCFSA_{FWP}$ has positive LE values along $r \in (0, 4]$ as shown in Figure 4 (k). LE is a quantifier for sensitivity to initial conditions. In other words, it is used to measure the rate of divergence of two initial conditions that start off infinitesimally close to one other after n iterations. Two input message blocks with a difference of just one bit can be viewed as two initial conditions. The small difference between the two blocks will grow with the increasing number of $DCFSA_{FWP}$ iterations, resulting in distinct chaotic points with a large difference. This will then result in two entirely different hash values, implying that the hash function is strongly dependent on the $DCFSA_{FWP}$ map in generating hash values.

C. DIFFUSION AND CONFUSION TEST

Diffusion and confusion properties are vital for a hash function in order to disperse message bits throughout a hash value and also obscures any relationship between message and hash value. In other words, good diffusion means each bit of an input message has an equal effect on the overall hash value. Meanwhile, having a good confusion property means that any slight change to the input message or key should lead to approximately half of the hash bits being flipped at randomly distributed binary locations [43]. The following steps are used to perform confusion and diffusion test:

1) Calculate the hash value, H_1 of a randomly selected message M.
2) Flip one random bit in M to obtain M'.
3) Calculate the hash value, H_2 of M'.
4) Note the number of changed bits, B between H_1 and H_2.
5) Repeat the experiment N times.

The following statistical tests are used to quantify the confusion and diffusion property of a hash algorithm [2], [27], [29]:

- 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 $\triangle P = \sqrt{\frac{1}{N-1}\sum_1^N (\frac{B_i}{128} - \frac{\bar{B}}{128})} \times 100\%$

The experiment is performed for $N = 512, 1024, 2048$ and 10000 and tabulated in Table2. The theoretical values of $\bar{B}$ and P are 64 and 50% respectively. Low values of $\triangle B$ and $\triangle P$ should be minimized to achieve good diffusion and confusion properties. The proposed hash function has a $\bar{B}$ that is extremely close to the theoretical value of 64. The values of the standard variances are also low, indicating a strong diffusion and confusion properties. Table 3 shows a comparison between the proposed function and other recently proposed chaos-based hash functions. The proposed hash function is at least on par or outperforms its peers in terms of achieving near-ideal results. Additionally, Figure 7 shows the frequency of the mean changed bit numbers. The normally distributed histogram centered at 64 bits again implies desirable diffusion and confusion properties.

TABLE 2. Statistical results for $N = 512$, 1024, 2048, 10,000 and 128-bit hash.

	N			
Parameters	512	1024	2048	10,000
$\bar{B}$	64.087	63.956	63.967	64.064
$P\%$	50.07	49.92	49.96	50.06
$\triangle B$	5.231	5.305	5.519	5.558
$\triangle P$	4.452	4.417	4.445	4.468
B_{min}	47	48	45	44
B_{max}	81	83	84	84

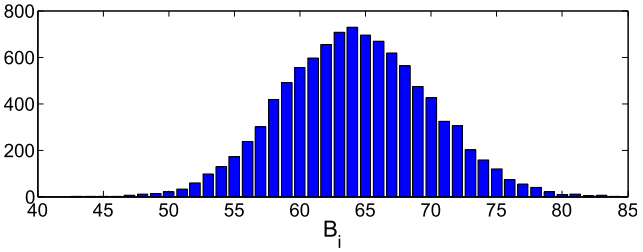

FIGURE 7. Distribution of changed bit number B_i.

D. COLLISION ANALYSIS

One of the design goals of hash functions is collision resistance. When two different input messages have the same hash value, a collision is said to have occurred. In this section, we analyze the susceptibility of the proposed hash function to collisions. Hash values are calculated for two different input messages which differ in only one bit. The two resulting hash values are recorded as ASCII characters and then compared. The number ASCII characters in the same position that are equal is considered a *hit*. Let ω denote the number of hits ($\omega = 0, 1, 2, \ldots, s$) whereas $W_N(\omega)$ denotes the number of times those specific number of hits occur after N experiments.

TABLE 3. Comparison of diffusion and confusion characteristics $N = 2048$.

Hash Function	$\bar{B}$	$P(\%)$	$\triangle B$	$\triangle P$
Proposed Function	**63.967**	**49.96**	**5.51**	**4.44**
Ref. [44] (Structure 1)	64.05	50.03	5.65	4.41
Ref. [44] (Structure 2 - $r = 8$)	63.88	49.91	5.66	4.42
Ref. [44] (Structure 2 - $r = 24$)	63.90	49.92	5.60	4.37
Ref. [45]	63.94	49.95	5.61	4.38
Ref. [46]	64.43	50.34	5.99	4.68
Ref. [27]	64.27	50.21	5.59	4.36
Ref. [23]	63.87	49.90	5.58	4.36
Ref. [26]	64.10	50.08	5.58	4.36
Ref. [47]	64.12	50.09	5.63	4.41
Ref. [48]	63.99	49.99	5.66	4.40
Ref. [49]	64.01	50.01	5.66	4.26
Ref. [50]	62.65	48.94	5.48	4.28
Ref. [25]	63.94	49.95	5.69	4.44
Ref. [24]	63.89	49.92	5.77	4.51
Ref. [51]	64.03	50.02	5.60	4.40
Ref. [52]	64.07	50.06	5.74	4.48
Ref. [53]	64.17	50.14	5.78	4.51
Ref. [54]	64.18	50.14	5.59	4.36
Ref. [55]	62.84	49.09	5.63	4.40
Ref. [56]	63.57	49.66	7.43	5.80
Ref. [57]	64.15	50.12	5.77	4.51
Ref. [58]	64.43	49.96	5.57	4.51
Ref. [59] (MD5)	64.03	50.02	5.66	4.42

The theoretical values of $W_N(\omega)$ for different N can be calculated as

$$F = \begin{cases} W_N(\omega) = N \times \dfrac{s!}{\omega!(s-\omega)!} \times (\dfrac{1}{2^8})^{\omega} \times (1-\dfrac{1}{2^8})^{s-\omega} \\ \sum_{\omega=0}^{s} W_N(\omega) = W_N(0) + W_N(1) + \ldots + W_N(s) = N \end{cases} \tag{10}$$

where $s = \frac{128}{8} = 16$ (128 is the length of the hash value whereas 8 bits are required for ASCII representation). When $N = 10000$ the theoretical values are $W_N(0) = 9392.98$, $W_N(1) = 589.36$, $W_N(2) = 17.33$, $W_N(3) = 0.3172, \ldots, W_N(16) = 2.94 \times 10^{-35}$. The results of proposed scheme are $W_N(0) = 9401$, $W_N(1) = 575$, $W_N(2) = 24$ and $W_N(> 2) = 0$ which is visually depicted in Figure 8. For $N = 10000$, the proposed hash function has near-ideal results which implies strong collision resistance. The proposed hash function's collision resistance property is benchmarked against its peers in Table 4, depicting a performance that is at least on par with the others with respect to the ideal collision resistance values.

E. ABSOLUTE DIFFERENCE

Firstly, a pair of input messages and the obtained hash values are conducted similar to the experiment in Section IV-D. We compare the two hash values in terms of ASCII format. The absolute difference d can be calculated as

$$d = \sum_{i=1}^{16} |t(e_i) - t(e_i{}')| \tag{11}$$

where e_i and $e_i{}'$ are the ith ASCII hash values in the original and modified input massages respectively while the function $t()$ converts an ASCII character into its corresponding decimal value. The theoretical value of the mean value of the absolute difference is important to compare with the proposed hash function. H is a discrete uniform distribution when using ASCII code for 16 characters (128-bita hash length), it has range of 0 to 255. The mean value of a uniform distribution is half of the maximum value of this distribution. All possible characters are equal to $255 \times 16 = 2,040$. According to [24],with the assumption that the two distinct hash values are ideally uniform, the sum of d of these two hash values has to be equal to $\frac{2}{3}$ of the mean value of a uniform distribution. Therefore, the theoretical mean value of the absolute difference for two hash values is equal to $\frac{2}{3} \times 2,040 = 1,360$ for 128-bits length. In the proposed function, The mean value of the absolute difference for $N = 10,000$ experiments is 1,351.345, which is very close to theoretical value.

TABLE 4. 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	**9401**	**575**	**24**	**0**	**0**	**0**	**0**
Ref. [45]	9414	569	17	0	0	0	0
Ref. [27]	9554	404	42	0	0	0	0
Ref. [23]	9387	586	27	0	0	0	0
Ref. [48]	9401	580	19	0	0	0	0
Ref. [49]	9969	31	0	0	0	0	0
Ref. [25]	9431	557	12	0	0	0	0
Ref. [24]	9387	591	22	0	0	0	0
Ref. [57]	9359	626	15	0	0	0	0

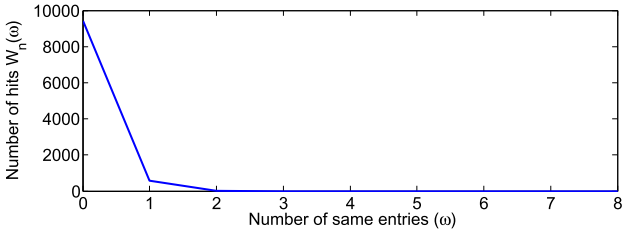

FIGURE 8. Distribution of number same ASCII codes at same location in hashes.

F. KEYSPACE ANALYSIS AND WEAK KEYS

Keyspace analysis is vital for a keyed hash function because it has a direct effect on its security. Generally, a keyspace should be large enough to resist the brute force attacks. When used as a keyed hash, keyspace calculation for the proposed hash function takes into consideration six chaotic maps, which have six initial conditions and six control parameters in total. The initial conditions are assigned to the buffer values of each machine state. The key consist of 12 variables that can calculated under floating point or fixed-point number. For initial conditions, the phase space of the three chaotic maps is between 0 and 1 under a bit precision of 2^{-52} bits. In this case, $U(0, 52)$ is used and the keyspace for initial conditions is $2^{52\times6} = 2^{312}$. For the control parameters, six control parameters have the same chaotic range of $(0, 4)$. To calculate the bit precision of these six parameters, $U(2, 50)$ is used to provide the chaotic parameter range. Then, the keyspace for control parameters is $2^{52\times6} = 2^{312}$.

TABLE 5. Comparison of keyspace values.

Algorithm	Keyspace
Proposed Function	2^{572}
Ref. [28]	$2^{383.9}$
Ref. [62]	2^{199}
Ref. [30]	2^{128}
Ref. [27]	2^{106}
Ref. [61]	$> 2^{128}$
Ref. [60]	2^{192}

When calculating keyspace, it is vital to take into consideration the existence of weak keys when it comes to chaos-based cryptographic algorithms. Unfortunately, it has not been addressed in many chaos-based cryptographic proposals [37]. A weak key can lead to a trivially constructed collision. In the case of the proposed hash function, initializing $b_1, b_2, \ldots b_5$ to zero must be avoided, regardless of the initial value of b_0. Otherwise, a collision can be constructed by first identifying a 312-bit message block, M_{col} whose first 52 bits combined with b_0 via modular addition will result in a final value of 0. This value of 0 will then be propagated throughout the DCFSA regardless of the number of iterations, producing a hash value of all zeroes. Other 312-bit message blocks consisting of all zeroes can be appended to M_{col} to generate collisions. This collision attack can be avoided by restricting the use of these weak keys, the number of which can be calculated as $2^{52} + (5 \times 52) \approx 2^{52}$. Thus, the keyspace of the proposed hash function is $2^{312+312-52} = 2^{572}$ which is large enough to resist estimation by brute force attacks. Furthermore, Table 5 shows the comparison of keyspace of the proposed hash function with other hash functions [27], [30], [60]–[62].

If the proposed hash function is used in keyed mode, its secret key comprises of the chaotic map initial conditions and control parameters. The initial conditions are used to initialize the state buffers prior to being modified by message bits. Iterating the DCFSA then diffuses the effect of the key bits to every machine state. On the other hand, the key bits used as control parameters influence the chaotic behavior of each individual 1D map, which indirectly affects the modification of each state buffer and the resulting hash value. In short, the entire secret key plays a strong role in producing hash values, whereby a small change to the key will lead to an entirely different hash value.

G. RESISTANCE TO BIRTHDAY ATTACK

The birthday attack is a generic attack that randomly selects half of the input possibilities in order to get a 50% chance of collision. Theoretically, an attacker only needs to test $n^{\frac{1}{2}}$ randomly selected possibilities of an n-bit hash value to discover a strong collision. In the proposed hash function, the hash length is 128 bits, whereby probability of finding a collision will be at least 2^{64}, which is still impractical by today's standard. In addition, the proposed hash function is scalable, whereby its hash length can be trivially extended due to the $DCFSA_{FWP}$ structure that has been employed. Thus, the proposed hash function is deemed secure against the birthday attack in the current attack model, excluding quantum models.

TABLE 6. Hashing speed comparison.

Algorithm	Speed (Gbps)
Proposed Function	**0.458**
Ref. [29]	0.433
Ref. [28]	0.214
Ref. [23]	0.425
Ref. [31]	0.182

H. FLEXIBILITY

One of the strengths of the proposed hash function is its flexibility to accommodate various hash lengths without significant computational overhead. One would just need to repeat the hash generation step $\frac{H}{32}$ times to generate a H-bit hash value. In a way, the DCFSA behaves slightly similar to a sponge construction, whereby message blocks are first absorbed by the DCFSA before being squeezed out as parts of the hash value. The proposed hash function also has the flexibility to be a keyed or unkeyed hash function. When used as an unkeyed hash function, the initial conditions specified in Section 4 would be used as constants. The proposed hash function also can be modified easily to produce different hash functions with different properties. Several possibilities are listed below:

- Increasing or decreasing the number of machine states of the $DCFSA_{FWP}$ to achieve different trade-offs in terms of security and efficiency.
- Assigning different chaotic maps or a combination of different chaotic maps to each of the machine states in $DCFSA_{FWP}$.
- Assigning different perturbation or chaotificiation methods to each of the transitions between the machine states in $DCFSA_{FWP}$.
- Increasing the number of machine states and chaotic maps of the $DCFSA_{FWP}$ to increase its keyspace.

I. SPEED ANALYSIS

One of the requirements of a cryptographic hash function is hashing speed. Hence, many researchers introduce new designs that focus on improving hashing efficiency. Table 6 shows the comparison between the proposed hash function with its peers [23], [28], [29], [31]. For a fair comparison, all of the hash functions were implemented on the same machine using Matlab R2012b on a Intel Core i5-560M @ 2.67GHz Processor, 8GB RAM and Windows 7 operating system. The proposed hash function is found to be more efficient than the rest.

The hash length in the proposed DCFSA hash algorithm is flexible due to how the DCFSA map is utilized (similar to the squeezing process of a cryptographic sponge). Generating larger hash values will have minimal effect on the overall performance because it only involves several

TABLE 7. Performance comparison of diffusion, confusion and collision characteristics.

Hash Function	Diffusion and Confusion, N=2048	Collision, N=10000		
	$\bar{B}$	$W_N(0)$	$W_N(1)$	$W_N(2)$
Ideal value	*64*	*9393*	*589*	*17*
Proposed Function	**63.96**	**9401**	**575**	**24**
Ref. [29]	64.24	9370	598	32
Ref. [28]	64.095	9396	583	21
Ref. [23]	63.87	9387	586	27
Ref. [27]	64.27	9554	404	42
Ref. [31]	64.17	9495	505	0
Ref. [48]	63.99	9401	580	19
Ref. [49]	64.01	9969	31	0
Ref. [25]	63.94	9431	557	12
Ref. [24]	63.89	9387	591	22

TABLE 8. Performance comparison of diffusion, confusion and mean value of absolute difference with secure hash algorithms (SHA).

	Proposed Function	SHA2-256	SHA3-256
$\bar{B}$	127.976	127.968	128.019
$P\%$	49.983	50.084	50.043
$\triangle B$	8.104	8.108	8.079
$\triangle P$	3.115	3.107	3.113
B_{min}	99	100	155
B_{max}	156	100	155
d	2738	2642	2708

additional iterations of the DCFSA map, which only iterates one 1D chaotic map each time. Each iteration produces 32 bits of hash values, thus even producing a 1024-bit hash value would only require what is essentially 32 iterations of a 1D map.

V. COMPARISON WITH OTHER ALGORITHMS

A comparison with other chaos-based hash functions [24], [25], [27]–[29], [31], [48], [49] is performed based on diffusion, confusion and collision analysis. The results tabulated in Table. 7 indicates that the performance of the proposed DCFSA hash function is comparable with existing work, which all achieve near-ideal statistical results.

Table 8 compares the proposed hash function with SHA-256 and SHA-3-256. In this comparison, we generate hash values of 256-bit length for a fair comparison in terms of diffusion, confusion, and absolute difference metrics. The proposed DCFSA hash function produces near-ideal results that are similar to SHA.

VI. DISCUSSION

Due to the properties of the *DCFSA* method, the resulting chaos-based hash function has uniformly distributed hash values with minimal chance of collisions. Each round in the hashing process takes 312 bits, divied into six blocks, and each chaotic machine state produces one chaotic point that is use to determine the state transition for the next chaotic map. The final result is generated by a ''squeezing'' process that has allows hash values of flexible lengths to be generated. The $DCFSA_{FWP}$ configuration was used because it has better chaotic performance as compared to the other *DCFSA* configurations. However, it needs two operations to generate chaos: the first one perturbs the control parameter and whereas the second perturbs the chaotic point. Therefore, if a *DCFSA* configuration that has fewer multiplication operations can be found, it could potentially improve the performance of the proposed algorithm even further.

In terms of security evaluation, statistical-based experiments have been the de facto methods when it comes to evaluating chaos-based hash functions. The same set of statistical-based experiments are also used to analyze the proposed hash function. The proposed hash function managed to produce near-ideal results for all test categories. However, near-ideal statistical values do not indicate that a hash function is resistant to advanced cryptanalytic attacks such as pseudo-preimage [63] and semi-free start collision attacks [64]. These attacks are generally performed independently, separate from the original hash function proposal. All chaos-based hash functions in recent literature have not taken these advanced attacks into consideration. One of the reasons for this is the use of floating point operations which complicates the use of these cryptanalytic techniques for security analysis. Hence, using fixed-point numbers in the design of new hash functions is facilitates future studies of its security aspects under these cryptanalytic analyses.

VII. CONCLUSION

In this paper, a new hash function based on deterministic chaotic finite state automata is proposed using fixed-point representation. Out of its various configurations, we select $DCFSA_{FWP}$ with six machine states and three simple chaotic maps to construct the hash function. Two forms of perturbations triggered by the message block, chaotic points, and threshold values are used to create dynamic changes that increase the randomness and sensitivity of the hash values. The proposed $DCFSA_{FWP}$-based hash function is evaluated in terms of various security metrics such as the hash value distribution, sensitivity to small changes of the message, confusion and diffusion properties, robustness against birthday attacks, absolute difference, keyspace and weak key analysis, collision tests, analysis of speed, and flexibility. Findings demonstrate that the proposed function has near-ideal statistical properties. By avoiding an overly complex construction and by using fixed point representation, the proposed hash function has a high degree of cryptanalytic analyzability from the binary point of view. Comparisons to the current state-of-the-art in chaos-based hash functions and secure hash algorithms show that the proposed hash function is at least on par, or outperforms its peers in multiple aspects such as security and efficiency. In future work, we will be looking into providing formal, mathematical descriptions for the behaviour and properties of the proposed hash function.

CONFLICT OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] C. S. Chum, C. Jun, and X. Zhang, "Implementation of randomize-then-combine constructed hash function," in *Proc. 23rd Wireless Opt. Commun. Conf. (WOCC)*, May 2014, pp. 1–6.

[5] M. Adeli and H. Liu, "Secure network coding with minimum overhead based on hash functions," *IEEE Commun. Lett.*, vol. 13, no. 12, pp. 956–958, Dec. 2009.

[6] 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.

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

[8] T. Salman, M. Zolanvari, A. Erbad, R. Jain, and M. Samaka, "Security services using blockchains: A state of the art survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 858–880, 1st Quart., 2019.

[9] 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. [Online]. Available: https://www.win.tue.nl/hashclash/rogue-ca/

[10] 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.

[11] 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-web-encryption-shattered

[12] A. Cilardo and N. Mazzocca, "Exploiting vulnerabilities in cryptographic hash functions based on reconfigurable hardware," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 5, pp. 810–820, May 2013.

[13] J. Lathrop, "Cube attacks on cryptographic hash functions," M.S. thesis, Rochester Inst. Technol., Rochester, NY, USA, 2009. [Online]. Available: https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=1653&context=theses

[14] R. F. Kayser, "Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family," *Fed. Register*, vol. 72, no. 212, p. 62, 2007.

[15] M. J. Dworkin, "SHA-3 standard: Permutation-based hash and extendable-output functions," Nat. Inst. Sci. Technol., Gaithersburg, MD, USA, Tech. Rep. 202, 2015.

[16] 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, Feb. 2019.

[17] D. Saha, S. Kuila, and D. R. Chowdhury, "Symsum: Symmetric-sum distinguishers against round reduced SHA3," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 1, pp. 240–258, 2017.

[18] 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.

[19] 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 *Proc. Int. Conf. Sel. Areas Cryptogr.* Cham, Switzerland: Springer, 2016, pp. 317–337.

[20] J. S. Teh, M. Alawida, and J. J. Ho, "Unkeyed hash function based on chaotic sponge construction and fixed-point arithmetic," *Nonlinear Dyn.*, vol. 100, pp. 713–729, Feb. 2020.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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, Jan. 2013.

[26] 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. 7, Jun. 2017, Art. no. 1750106.

[27] 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.

[28] M. Ahmad, S. Singh, and S. Khurana, "Cryptographic one-way hash function generation using twelve-terms 4D nonlinear system," *Int. J. Inf. Technol.*, pp. 1–9, May 2018, doi: 10.1007/s41870-018-0199-8.

[29] 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.

[30] 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.

[31] 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.

[32] D. Arroyo, G. Alvarez, and V. Fernandez, "On the inadequacy of the logistic map for cryptographic applications," 2008, *arXiv:0805.4355*. [Online]. Available: http://arxiv.org/abs/0805.4355

[33] G. Alvarez, J. M. Amigó, D. Arroyo, and S. Li, "Lessons learnt from the cryptanalysis of chaos-based ciphers," in *Studies in Computational Intelligence*. Berlin, Germany: Springer, 2011, pp. 257–295.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] W. Guo, X. Wang, D. He, and Y. Cao, "Cryptanalysis on a parallel keyed hash function based on chaotic maps," *Phys. Lett. A*, vol. 373, no. 36, pp. 3201–3206, Aug. 2009.

[39] M. Alawida, A. Samsudin, J. S. Teh, and W. H. Alshoura, "Deterministic chaotic finite-state automata," *Nonlinear Dyn.*, vol. 98, pp. 2403–2421, Oct. 2019.

[40] 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.

[41] C. Li, B. Feng, S. Li, J. Kurths, and G. Chen, "Dynamic analysis of digital chaotic maps via state-mapping networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 6, pp. 2322–2335, Jun. 2019.

[42] 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.

[43] B. Coskun and N. Memon, "Confusion/diffusion capabilities of some robust hash functions," in *Proc. 40th Annu. Conf. Inf. Sci. Syst.*, Mar. 2006, pp. 1188–1193.

[44] 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. 11, pp. 2137–2161, Feb. 2019.

[45] 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, pp. 17973–17994, Jan. 2019.

[46] 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. Sci. Tech. Sci.*, vol. 67, no. 3, pp. 557–569, 2019.

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

[48] 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.

[49] 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.
[50] 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.
[51] H. Bo, L. Peng, P. Qin, and L. Zhaolong, "A method for designing hash function based on chaotic neural network," in *Proc. 1st Int. Workshop Cloud Comput. Inf. Secur.*, 2013.
[52] 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.
[53] 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.
[54] M. Nouri, A. Khezeli, A. Ramezani, and A. Ebrahimi, "A dynamic chaotic hash function based upon circle chord methods," in *Proc. 6th Int. Symp. Telecommun. (IST)*, Nov. 2012, pp. 1044–1049.
[55] N. Jiteurtragool, P. Ketthong, C. Wannaboon, and W. San-Um, "A topologically simple keyed hash function based on circular chaotic sinusoidal map network," in *Proc. 15th Int. Conf. Adv. Commun. Technol. (ICACT)*, 2013, pp. 1089–1094.
[56] Y. Li, D. Xiao, S. Deng, Q. Han, and G. Zhou, "Parallel hash function construction based on chaotic maps with changeable parameters," *Neural Comput. Appl.*, vol. 20, no. 8, pp. 1305–1312, Nov. 2011.
[57] 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.
[58] H. Zhang, X.-F. Wang, Z.-H. Li, and D.-H. Liu, "One way hash function construction based on spatiotemporal chaos," *Acta Phys. Sinica*, vol. 54, no. 9, pp. 4006–4011, 2005.
[59] R. Rivest, *The MD5 Message-Digest Algorithm*, document RFC 1321, 1992.
[60] A. Kanso and M. Ghebleh, "A structure-based chaotic hashing scheme," *Nonlinear Dyn.*, vol. 81, nos. 1–2, pp. 27–40, Jul. 2015.
[61] 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. 22, pp. 905–919, Jul. 2017.
[62] 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
[63] Y. Sasaki and K. Aoki, "Finding preimages in full MD5 faster than exhaustive search," in *Advances in Cryptology*. Berlin, Germany: Springer, 2009, pp. 134–152.
[64] F. Mendel, T. Nad, and M. Schläffer, "Improving local collisions: New attacks on reduced SHA-256," in *Advances in Cryptology*. Berlin, Germany: Springer, 2013, pp. 262–278.

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/cryptography and cybersecurity from the School of Computer Sciences, University Sains Malaysia, in 2020. His research interests include chaotic system, chaos-based applications, multimedia security, blockchain, and cryptography.

JE SEN TEH received the B.Eng. degree (Hons.) in electronics from Multimedia University, Malaysia, in 2011, the M.Sc. degree in computer science from Universiti Sains Malaysia, in 2013, and the Ph.D. degree from the School of Computer Sciences, Universiti Sains Malaysia, in 2017. 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.

DAMILARE PETER OYINLOYE received the B.Sc. (Hons.) and M.Sc. degrees in computer science from Kwara State University, Malete, Nigeria, in 2013 and 2016, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Sciences, Universiti Sains Malaysia. He is also a Lecturer and a Researcher with Kwara State University. His research interests include blockchain, data privacy and security, cryptography, and residue number systems.

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.

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. He has been working as an Assistant Professor with the Department of Computer Engineering, Jamia Millia Islamia, New Delhi, since 2011. He has published about 70 research articles in refereed journals and conference proceedings of international repute. His areas of research interest include multimedia security, chaos-based cryptography, cryptanalysis, and optimization techniques.

RAMI S. ALKHAWALDEH received the B.S. degree in computer information systems from Yarmouk University, Irbid, Jordan, in 2007, the M.Sc. degree in computer information system from The University of Jordan, Aqaba, Jordan, in 2010, and the Ph.D. degree in computing science from Glasgow University, U.K., in 2017. From 2010 to 2012, he was a Lecturer with The University of Jordan. Since February 2016, he has been an Assistant Professor with the Computer Information Systems Department, The University of Jordan. His research interests include artificial intelligence, machine learning, information retrieval, VOIP, and wireless networks.