INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2018, VOL. 64, NO. 4, PP. 457-465
Manuscript received April 29, 2018; revised October, 2018. DOI: 10.24425/123546

SHAH: Hash Function Based on Irregularly Decimated Chaotic Map

Mihaela Todorova, Borislav Stoyanov, Krzysztof Szczypiorski, and Krasimir Kordov

***Abstract*—We present a new hash function based on irregularly decimated chaotic map, in this article. The hash algorithm called SHAH is based on two Tinkerbell maps filtered with irregular decimation rule. We evaluated the novel function using distribution analysis, sensitivity analysis, static analysis of diffusion, static analysis of confusion, and collision analysis. The experimental data show that SHAH satisfied valuable level of computer security.**

***Keywords*—Hash function, Chaotic functions, Shrinking decimation rule, Pseudo-random number generator**

I. INTRODUCTION

DURING recent decades, with the dynamic development of computer science and information technologies, network security tools are becoming increasingly important.

Decimation sequences play a big part in the area of basic cryptographic primitives. The output bits are produced by applying a threshold function into a sequence of numbers. The resulting decimation sequence has good randomness properties. In [6], two linear feedback shift registers (LFSRs) and threshold function are used to create novel output of pseudo-random bits. An algorithm of a pseudo-random generation based on a single LFSR is proposed in [21]. In [12], a class of irregularly decimated keystream schemes, based on 1-D piecewise map is presented. Pseudo-random sequences constructed from two Chebyshev maps, filtered by decimation function are presented in [25]. In [4], [9], [14], [15], [17], [24], [26]–[28] new pseudo-random bit generation techniques, software applications based on chaotic maps, and studies about sets are presented.

A cryptography hash function is a one-way algorithm used for compression of a plain text of arbitrary length into a secret binary sequence of fixed-size. The hash function provides the necessary security in authentication methods and digital signature algorithms.

Novel chaos-based hash algorithm, which uses m-dimensional Cat map, is proposed in [16] and improved in [7]. Another hash function based on a Cat map, is defined in [10]. Based on a chaotic system, a hash construction which has high performance is designed in [22]. A chaotic look-up table based on a tent map is used to design a novel 128-bit hash function in [18]. A hash algorithm based on a tent map is constructed in [13]. In [11], a 2D generalized Cat map is used to present an improved hash function.

In [31], a circular-shift-based chaotic hash function, is constructed. A hash function by low 8-bit of 8D hyperchaotic system iterative outputs is proposed in [19]. In [1], an algorithm for generating secure hash codes using a number of chaotic maps is designed.

The aim of the article is to construct a new hash algorithm based on irregularly decimated chaotic map.

In Section II we propose a novel pseudo-random bit scheme based on two Tinkerbell maps filtered with shrinking rule. In Section III we present the novel hash function SHAH and detailed security analysis is given. Finally, the last part concludes the article.

II. PSEUDO-RANDOM BIT GENERATOR BASED ON IRREGULARLY DECIMATED CHAOTIC MAP

The work presented in this section was motivated by recent developments in chaos-based pseudo-random generation [8], [29], [30], and with respect of [6].

A. *Proposed Pseudo-random Bit Generation Algorithm*

The Tinkerbell map [2] is given by the following function:

$$\begin{aligned} x_{n+1} &= x_n^2 - y_n^2 + c_1 x_n + c_2 y_n \\ y_{n+1} &= 2 x_n y_n + c_3 x_n + c_4 y_n \ . \end{aligned} \tag{1}$$

The map depends on the four paremeters c_1, c_2, c_3, and c_4. The Tinkerbell map with different values of the parameters is illustrated in Figure 1.

The shrinking generator [6] uses two sequences of pseudo-random bits (**a** and **s**) to create a third output sequence (**z**) of pseudo-random bits which includes those bits a_i for which the corresponding s_i is 1. Other bits from the first sequence are decimated.

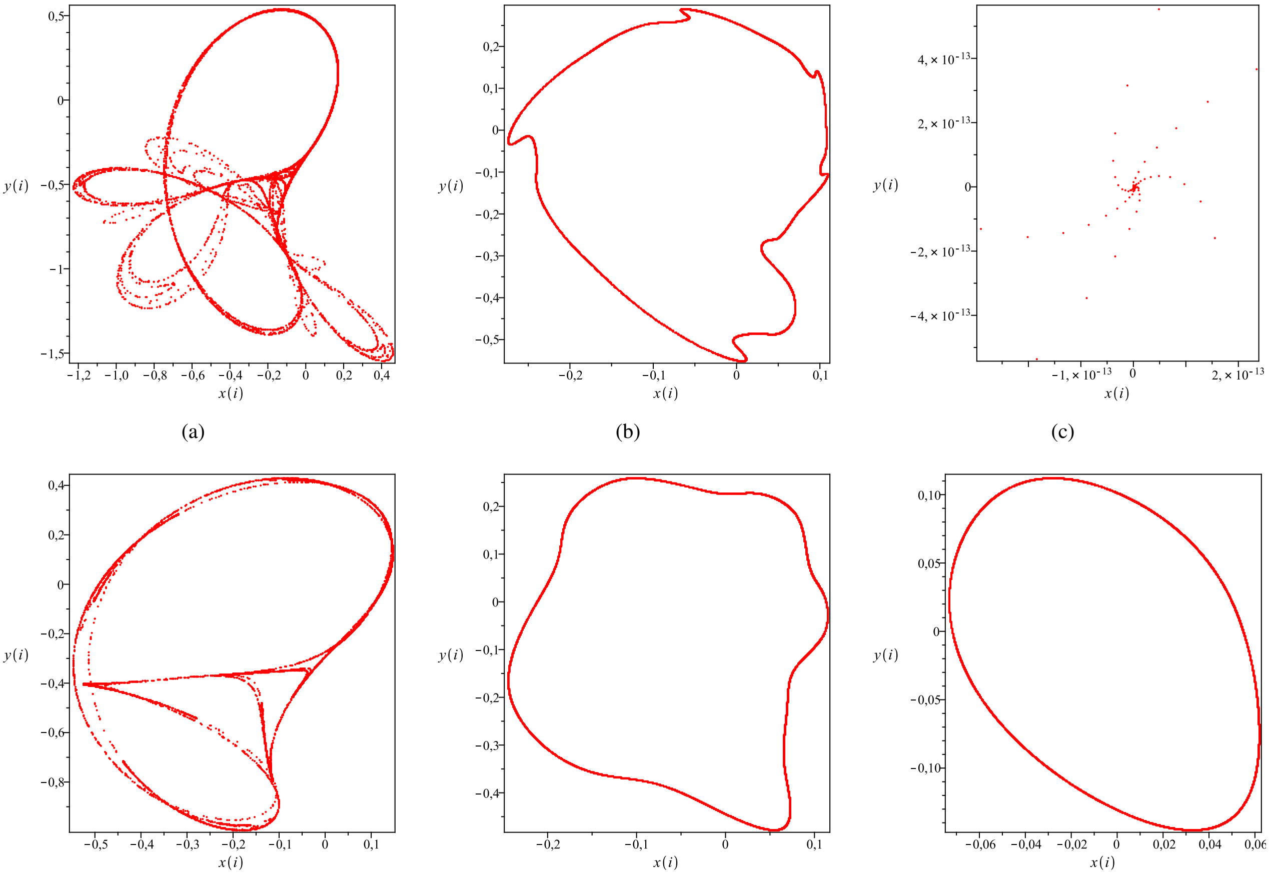
We propose a novel pseudo-random number scheme which irregularly decimates the outputs of two Tinkerbell functions by using the shrinking rule [6]. We used the following parameters $c_1 = 0.9$, $c_2 = -0.6013$, $c_3 = 2.0$ and $c_4 = 0.50$. The novel generator is based on the following equations:

$$\begin{aligned} x_{1,n+1} &= x_{1,n}^2 - y_{1,n}^2 + c_1 x_{1,n} + c_2 y_{1,n} \\ y_{1,n+1} &= 2 x_{1,n} y_{1,n} + c_3 x_{1,n} + c_4 y_{1,n} \\ x_{2,m+1} &= x_{2,m}^2 - y_{2,m}^2 + c_1 x_{2,m} + c_2 y_{2,m} \\ y_{2,m+1} &= 2 x_{2,m} y_{2,m} + c_3 x_{2,m} + c_4 y_{2,m} \ , \end{aligned} \tag{2}$$

This work is supported by the Scientic research fund of Konstantin Preslavski University of Shumen under the grant No. RD-08-121/06.02.2018 and by European Regional Development Fund and the Operational Program ”Science and Education for Smart Growth” under contract UNITe No. BG05M2OP001-1.001-0004-C01 (2018-2023).

M. Todorova, B. Stoyanov, and K. Kordov are with the Department of Computer Informatics, Konstantin Preslavsky University of Shumen, 9712 Shumen, Bulgaria (e-mails: mihaela.todorova@shu.bg, borislav.stoyanov@shu.bg, krasimir.kordov@shu.bg).

Krzysztof Szczypiorski is with Warsaw University of Technology, Warsaw, Poland; Cryptomage SA, Wroclaw, Poland (e-mail: ksz@tele.pw.edu.pl).



(a) (b) (c)

(d) (e) (f)

Fig. 1. The Tinkerbell map of Eq.(1). This is a plot of (x_i, y_i), for $i = 0..100000$. (**a**) Tinkerbell map with $c_1 = 0.9$, $c_2 = -0.6013$, $c_3 = 2.0$, and $c_4 = 0.50$, (**b**) Tinkerbell map with $c_1 = 0.3$, $c_2 = -0.6013$, $c_3 = 2.0$, and $c_4 = 0.50$, (**c**) Tinkerbell map with $c_1 = 0.9$, $c_2 = -0.6013$, $c_3 = 2.0$, and $c_4 = -0.4$, (**d**) Tinkerbell map with $c_1 = 0.9$, $c_2 = -0.6013$, $c_3 = 2.0$, and $c_4 = 0.4$, (**e**) Tinkerbell map with $c_1 = 0.3$, $c_2 = -0.6013$, $c_3 = 2.0$, and $c_4 = 0.4$, and (**f**) Tinkerbell map with $c_1 = -0.3$, $c_2 = -0.6013$, $c_3 = 2.0$, and $c_4 = 0.5$.

where initial values $x_{1,0}, y_{1,0}, x_{2,0}$ and $y_{2,0}$ are used as a key.

Step 1: The starting points $x_{1,0}, y_{1,0}, x_{2,0}$ and $y_{2,0}$ of the two Tinkerbell functions from Eqs. (2) are setted up.

Step 2: The two functions from Eqs. (2) are iterated for M and N times, respectively.

Step 3: The computing of the Eqs. (2) continues, and $y_{1,n}$ and $y_{2,m}$ are filtered as follows:

$$\begin{aligned} a_i &= abs(mod(integer(y_{1,n} \times 10^9), 2)) \\ s_i &= abs(mod(integer(y_{2,m} \times 10^9), 2), \end{aligned} \qquad (3)$$

where $abs(x)$ returns the absolute value of x, $integer(x)$ returns the the integer part of x, truncating the value at the decimal point, $mod(x, y)$ returns the reminder after division.

Step 4: Apply the shrinking rule [6] to the values (a_i, s_i) and produce the output bit.

Step 5: Return to Step 3 until the output bits limit is attained.

The novel hash algorithm is coded in C++, using the input values: $x_{1,0} = -0.423555643379287$, $y_{1,0} = -0.762576287931311$, $M = N = 3500$, $x_{2,0} = -0.276976682878721$, and $y_{2,0} = -0.348339839900213$.

B. Initial Key Size Calculation

The initial key size is the set of all initial values of the pseudo-random bit generation steps. The proposed generator has four input parameters $x_{1,0}$, $y_{1,0}$, $x_{2,0}$, and $y_{2,0}$. According to [34], the computational precision of the 64-bit double-precision floating-point format is about 10^{-15}, thus the initial key size is more than 2^{199}. The proposed pseudo-random generator is secure against brute-force key size search [3]. Moreover, the starting iteration values M and N can be included as a part of the key size.

C. Statistical tests

To measure statistical randomness of the output produced by the proposed pseudo-random number algorithm, we used the NIST suite [5], DIEHARD suite [20], and ENT suite [32].

The NIST package Version 2.1.1 includes the following statistical tests of randomness: monobit, block-frequency, cumulative sums, runs, longest run of ones, rank, spectral, non overlapping templates, overlapping templates, universal, approximate entropy, random excursions, random excursion variant, serial one, serial two, and linear complexity.

For the NIST tests, we outputted 1000 different binary sequences of length 1,000,000 bits. The results from the tests

TABLE I
NIST TEST SUITE RESULTS

NIST test	P-value	Pass rate
Frequency (monobit)	0.869278	981/1000
Block-frequency	0.548314	985/1000
Cumulative sums (Reverse)	0.790621	983/1000
Runs	0.610070	990/1000
Longest run of Ones	0.439122	984/1000
Rank	0.467322	989/1000
FFT	0.058612	988/1000
Non-overlapping templates	0.519879	991/1000
Overlapping templates	0.510153	982/1000
Universal	0.159910	989/1000
Approximate entropy	0.616305	991/1000
Random-excursions	0.641892	588/594
Random-excursions Variant	0.495265	589/594
Serial 1	0.614226	989/1000
Serial 2	0.151190	985/1000
Linear complexity	0.620465	990/1000

are given in Table I. The minimum pass rate for each statistical test with the exception of the random excursion variant test is 980 for a sample size of 1000 sequences. The minimum pass rate for the random excursion (variant) test is 580 for a sample size of 594 sequences. The Tinkerbell map pseudo-random bit generator passed very good all the NIST tests.

The DIEHARD application [20] is a set of following tests: birthday spacings, overlapping 5-permutations, binary rank (31 x 31), binary rank (32 x 32), binary rank (6 x 8), bitstream, overlapping-pairs-sparse-occupancy, overlapping-quadruples-sparse-occupancy, DNA, stream count-the-ones, byte-count-the-ones, parking lot, minimum distance, 3D spheres, squeeze, overlapping sums, runs (up and down), and craps. The tests produce $P - values$, which should be uniform in [0,1), if the input stream contains pseudo-random numbers. The $P-values$ are obtained by $p = F(y)$, where F is the assumed distribution of the sample random variable y, often the normal distribution. The novel pseudo-random bit algorithm passed very well all DIEHARD tests, Table II.

The ENT software [32] includes 6 tests of pseudo-random streams: entropy, optimum compression, χ^2 distribution, arithmetic mean value, Monte Carlo value for π, and serial correlation coefficient. Streams of bytes are saved in files. We tested output of 125,000,000 bytes of the Tinkerbell function based pseudo-random number generation scheme. The novel pseudo-random bit generation algorithm passed successfully all ENT tests, Table III.

Based on the good test results, we can say that the proposed pseudo-random bit generation algorithm has satisfying good statistical properties and provides acceptable level of security.

TABLE II
DIEHARD TEST SUITE RESULTS

DIEHARD	P-value
Birthday spacings	0.513830
Overlapping 5-permutation	0.927974
Binary rank (31 x 31)	0.890892
Binary rank (32 x 32)	0.609788
Binary rank (6 x 8)	0.486987
Bitstream	0.662411
OPSO	0.618526
OQSO	0.445982
DNA	0.526710
Stream count-the-ones	0.299022
Byte count-the-ones	0.546796
Parking lot	0.574512
Minimum distance	0.115118
3D spheres	0.527506
Squeeze	0.678411
Overlapping sums	0.556561
Runs up	0.543542
Runs down	0.438540
Craps	0.272223

TABLE III
ENT TEST SUITE RESULTS

ENT tests	Results
Entropy	7.999998 bits per byte
Optimum compression	OC would decrease the file size by 0 %.
χ^2 distribution	For 125000000 samples is 278.28, and randomly would exceed this value 15.15 % of the time.
Arithmetic mean value	127.5015 (127.5 = random)
Monte Carlo π estim.	3.141354290 (error 0.01 %)
Serial correl. coeff.	0.000115 (totally uncorrelated = 0.0)

III. HASH FUNCTION BASED ON IRREGULARLY DECIMATED CHAOTIC MAP

A. Proposed Hash Function based on Irregularly Decimated Chaotic Map

In this section, we construct a keyed hash function named SHAH based on a irregularly decimated chaotic map. Let n be the bit length of the final hash code. The parameter n usually supports five bit lengths, 128, 160, 256, 512, and 1024 bits. We consider input message M' with arbitrary length.

The novel hash algorithm SHAH consists of the following steps:

Step 1: Convert the input string M to binary string using ASCII table.
Step 2: The input stream M is padded with a bit of one, and then append zero bits to obtain a message M' whose length is m, a multiple of n.

Step 3: The novel pseudo-random bit generation algorithm (Section II) based on two Tinkerbell functions filtered with shrinking rule is iterated many times, getting m bits, m-sized vector P.

Step 4: The m-sized vectors M' and P are combined in a new m-sized vector, N, using XOR operation.

Step 5: The vector N is split into p blocks, $N_1, N_2, ..., N_p$, each of length n and $m = np$ is the total length of the vector N.

Step 6: A temporary n-sized vector T is obtained by $T = N_1 \oplus N_2 \oplus \cdots \oplus N_p$.

Step 7: The bits from the temporary vector T are processed one by one sequentially. If the current bit t_i is 1 then update $t_i = t_i \oplus s$, where s is the next bit from the novel pseudo-random generator based on Tinkerbell function (Section II).

Step 8: Another n-sized temporary vector U is taken and all the elements are initialized to 0s.

Step 9: The bits from the vector T are processed again one by one sequentially. If the current bit t_i is 1, the vector U is XOR-ed with the next n bits from the novel pseudo-random generator based on Tinkerbell function (Section II). If the current bit t_i is 0, the matrix U is bitwise rotated left by one bit position.

Step 10: The final hash code is obtained by $H = T \oplus U$.

The designed SHAH algorithm is implemented in C++ programming language.

B. Distribution Analysis

In general, a typical property of a hash code is to be uniformly distributed in the compressed range. Note that the length of the hash code is set as 128. Simulation experiments are done on the following paragraph of input stream (http://shu.bg/about-us/history?language=en):

> *Konstantin Preslavsky University of Shumen has inherited a centuries-long educational tradition dating back to the famous Pliska and Preslav Literary School (10th c). Shumen University is one of the five classical public universities in Bulgaria it is recognized as a leading university that offers modern facilities for education, scientific researches and creative work.*

With the chosen input message, the SHAH hash code is calculated. The ASCII code distribution of the input message and the output hexadecimal hash code are shown in Fig. 2(a) and 2(b). Another input message with the same length but all of whitespaces, is generated. The ASCII code distribution of the whitespaced input message and the corresponding hexadecimal hash code are shown in Fig. 2(c) and 2(d). The SHAH hash plots, Fig. 2(b) and 2(d), are uniformly distributed in compress range even under exceptionally cases.

C. Sensitivity Analysis

To demonstrate the sensitivity of the proposed keyed hash function to the input message and security key space, hash algorithm tests have been performed under the following 9 cases:

C1: The input message is the same as the one in Section III-B;

C2: Change the first character *'K'* in the input message into *'k'*.

C3: Change the number *'10'* in the input message to *'11'*.

C4: Change the word *'School'* in the input message to *'school'*.

C5: Change the comma *','* in the input message to *'.'*.

C6: Add an whitespace at the end of the input message.

C7: Change the word *'recognized'* in the input message to *'recognize'*.

C8: Subtracts 1×10^{-15} from the input key value $x_{1,0}$.

C9: Adds 1×10^{-15} to the input key value $y_{2,0}$.

The respective 128-bit hash code in hexadecimal numeral system are the following:

C1: 8CE855A3026CCEE597C0965B5DB33096
C2: 8F0E33DA59B5B1114F9A1570EB466C24
C3: 11DEA1F51379EC2B429325D16FD5354C
C4: 6B834AC8D36B74EFAD0C6B8AAEA008BF
C5: 81704BC6412FF4E24AF09E570AB4D9DE
C6: A2B7D2EAC687D2953551AE2621720ADF
C7: E1D9A0BAA6184264481A25D08BEFF110
C8: 780378A8FE0011DBD81CE035414907F0
C9: C27AB518A87B8E3D6C46504814BF7940

The corresponding binary representation of the hash codes are illustrated in Fig. 3.

The result shows that the novel hash algorithm based on irregularly decimated chaotic map has high sensitivity to minimal changes to its security key space. Even tiny changes in secret keys or in input messages will lead to significant differences of hash codes.

D. Statistic Analysis of Diffusion and Statistic Analysis of Confusion

From a historical point of view, Shannon, with the publication in 1949 of his paper, *Communication Theory of Secrecy Systems* [23], introduced the idea of two methods for frustrating a statistical analysis of encryption algorithms: confusion and diffusion.

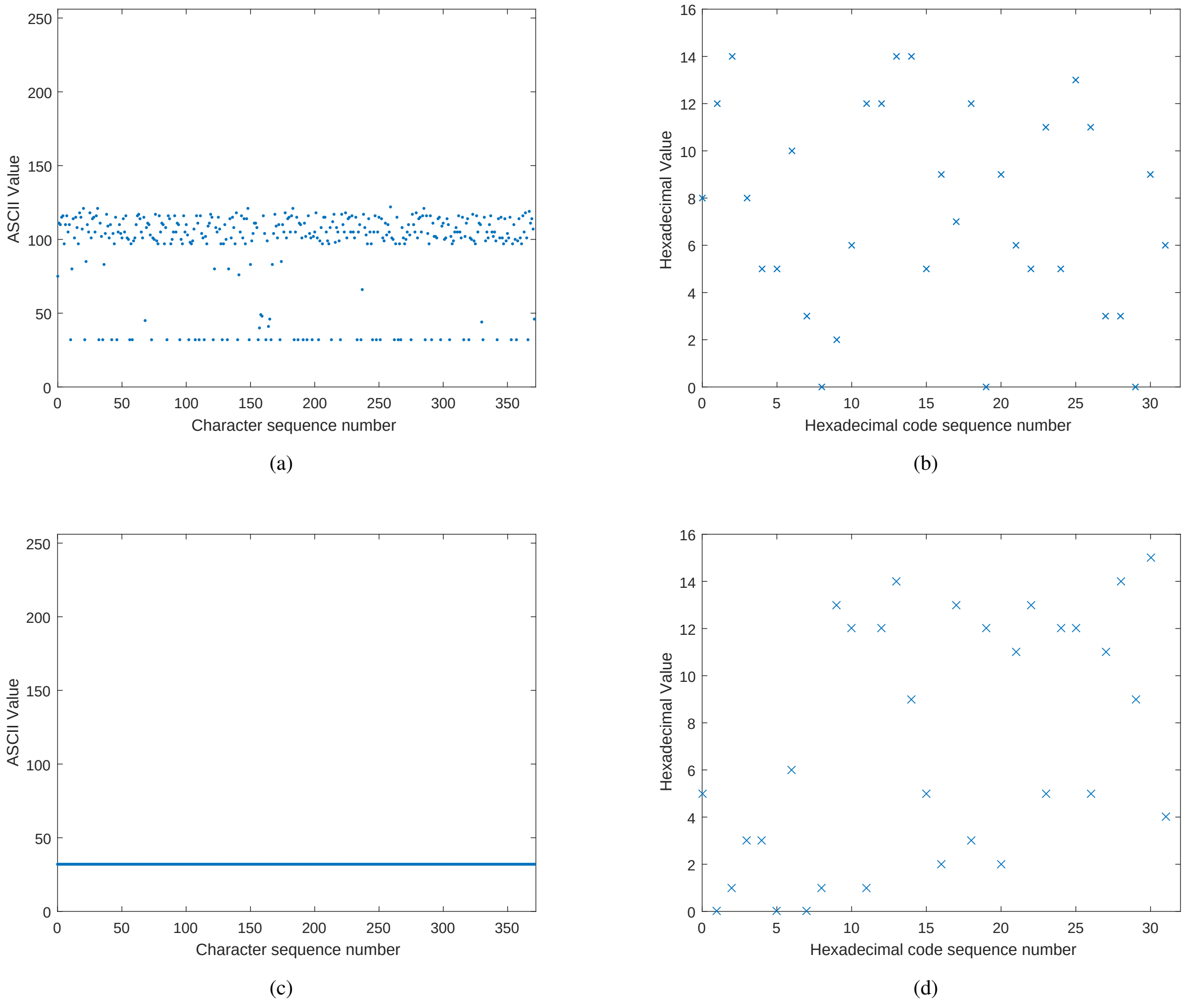
Confusion is intended to use transformations to hide the relationship between the plaintext and ciphertext, which means the relationship between the plaintext and the hash code is as complicated as possible.

Diffusion can propagate the change over the whole encrypted message, which means that the hash code is highly dependent on the input data. For a binary representation of the hash code, each bit can be only zero or one. Therefore, the ideal diffusion effect should be that any small changes in the starting values lead to a 50% changing probability of each bit of hash code [33].

Six statistics used here are [18]: minimum number of changed bits B_{min}, maximum number of changed bits: B_{max}, mean changed bit number $\bar{B}$, mean changed probability P, standard deviation of the changed bit number ΔB, and standard deviation ΔP.

They are defined as follows:

Minimum number of changed bits: $B_{min} = min(\{B_i\}_{i=1}^{N})$



(a) (b) (c) (d)

Fig. 2. Distribution of input string and output hash code.

Maximum number of changed bits: $B_{max} = max(\{B_i\}_{i=1}^{N})$

Mean changed bit number: $\bar{B} = \frac{1}{N}\sum_{i=1}^{N} B_i$

Mean changed probability: $P = \frac{\bar{B}}{n} \times 100\%$

Standard deviation of numbers of changed bits: $\Delta B = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(B_i - \bar{B})^2}$

Standard deviation: $\Delta P = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\frac{B_i}{n} - P)^2} \times 100\%$,

where N is the total number of tests and B_i is the number of changed bits in the i-th test (Hamming distance).

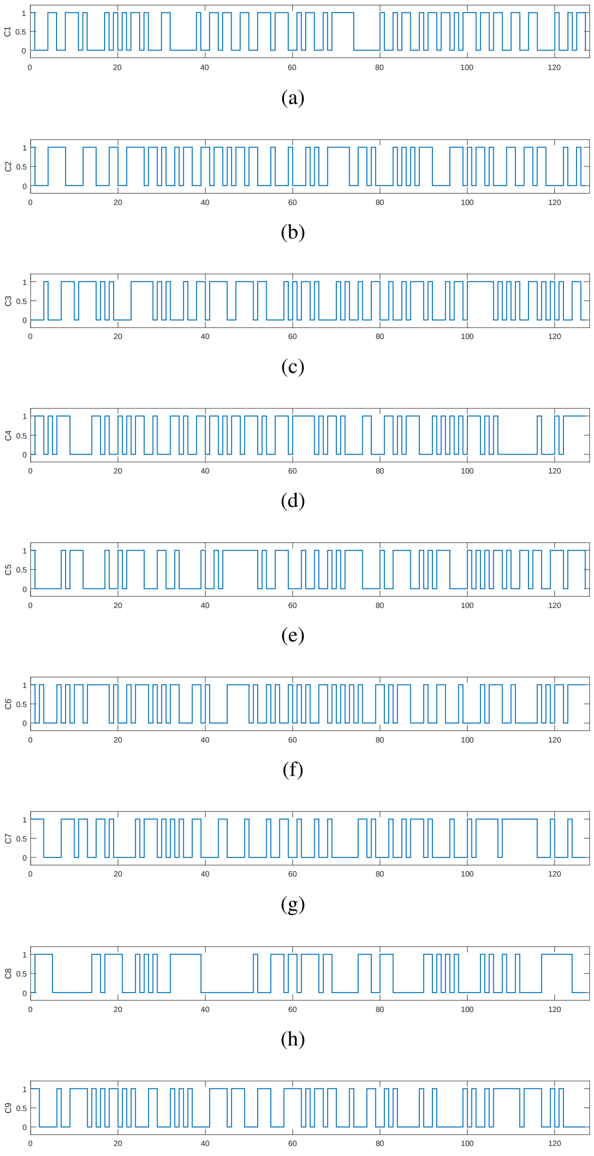
Two types of statistical tests are performed [10]: type A and type B. In the type A test, an input stream, of size $L = 50n$ is generated and its corresponding $n-$bit hash code is generated. Then, a new stream is computed by choosing a single bit at random from the input stream and modified to 0 if it is 1 or to 1 if it is 0. The n-bit hash code of the new stream is then compared with that of the original stream and the Hamming distance between the two hash codes is recorded as B_i. This is then repeated N times, where each time, a new original stream is chosen and one of its bits is randomly chosen and modified to 0 if it is 1 or to 1 if it is 0. Tables IV–VIII present results of these tests for $n = 128$, 160, 256, 512, and 1024.

TABLE IV
STATISTICS FOR 128-BIT HASH CODES, TYPE A TESTS.

	N=256	N=512	N=1024	N=2048	N=10,000
B_{min}	51	49	47	45	45
B_{max}	80	89	89	89	89
$\bar{B}$	64.5	63.998	64.11	63.99	64
$P(\%)$	50.39	49.99	50.08	49.99	50.01
ΔB	5.43	5.66	5.31	5.51	5.6
$\Delta P(\%)$	4.425	4.19	4.15	4.33	4.37

TABLE V
STATISTICS FOR 160-BIT HASH CODES, TYPE A TESTS.

	N=256	N=512	N=1024	N=2048	N=10,000
B_{min}	61	59	56	56	56
B_{max}	94	96	100	101	101
$\bar{B}$	74.31	79.74	79.99	80	79.92
$P(\%)$	49.57	49.84	49.99	50	49.95
ΔB	5.86	6.07	6.21	6.39	6.3
$\Delta P(\%)$	3.66	3.79	3.88	3.99	3.94



(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

(i)

Fig. 3. 128-bit hash codes of the input messages under nine different cases: (**a**) C1, (**b**) C2, (**c**) C3, (**d**) C4, (**e**) C5, (**f**) C6, (**g**) C7, (**h**) C8, and (**i**) C9.

TABLE VI
STATISTICS FOR 256-BIT HASH CODES, TYPE A TEST.

	N=256	N=512	N=1024	N=2048	N=10,000
B_{min}	103	102	102	102	92
B_{max}	148	152	156	161	162
$\bar{B}$	128.01	127.95	127.95	127.94	128.04
$P(\%)$	50	49.98	49.98	49.97	50.01
ΔB	8.04	8.1	7.99	7.94	7.99
$\Delta P(\%)$	3.14	3.16	3.12	3.1	3.12

TABLE VII
STATISTICS FOR 512-BIT HASH CODES, TYPE A TESTS

	N=256	N=512	N=1024	N=2048	N=10,000
B_{min}	214	214	214	214	212
B_{max}	287	287	287	293	302
$\bar{B}$	255.94	255.95	256.03	255.74	256.04
$P(\%)$	49.99	49.99	50.01	49.95	50
ΔB	12.73	11.84	11.48	11.44	11.33
$\Delta P(\%)$	2.48	2.31	2.24	2.23	2.21

TABLE VIII
STATISTICS FOR 1024-BIT HASH CODES, TYPE A TESTS.

	N=256	N=512	N=1024	N=2048	N=10,000
B_{min}	471	469	464	454	448
B_{max}	561	561	561	563	577
$\bar{B}$	513.7	512.73	512.25	511.79	511.97
$P(\%)$	50.16	50.07	50.02	49.98	49.99
ΔB	15.42	15.37	15.47	15.61	15.85
$\Delta P(\%)$	1.5	1.5	1.51	1.52	1.54

Comparing variables with most of these existing hash algorithms given in Table IX, the SHAH has small ΔB and ΔP values, respectively.

In the type B test, the original message M of size $L = 50n$ bits is generated at random and its corresponding n-bit hash code is computed. Then, a single bit of the original message is chosen, modified to 0 if it is 1 or to 1 if it is 0, and the hash code of the modified message is calculated. The two hash codes are compared, and the number of flipped bits is calculated as B_i. The same input stream is used for all N steps. Tables X–XIV list the results obtained in tests of type B for $n = 128, 160, 256, 512, 1024$, and different numbers of N.

Comparing the results with few chaos based hash algorithms given in Table XV, the SHAH has small ΔB and ΔP values, accordingly.

In Tables IV–XIV we can observe that both types of tests, the mean changed bit number $\bar{B}$ and the mean probability P are very close to the ideal values $n/2$ and 50%. These

TABLE IX
COMPARISON OF STATISTICS 128-BIT HASH CODES AND N=10,000, TYPE A TESTS

	B_{min}	B_{max}	$\bar{B}$	$P(\%)$	ΔB	$\Delta P(\%)$
SHAH	45	89	64	50.01	5.6	4.37
Ref. [10]	45	84	63.94	49.95	5.64	4.41
Ref. [11]	44	84	64	50	5.65	4.41
Ref. [13]	46	82	64.15	50.12	5.74	4.48
Ref. [19]	44	84	63.95	49.96	5.62	4.39
Ref. [33]	42	83	63.986	49.988	5.616	4.388

TABLE X
STATISTICS FOR 128-BIT HASH CODES, TYPE B TESTS.

	N=256	N=512	N=1024	N=2048	N=50×128
B_{min}	53	49	48	43	43
B_{max}	78	83	83	83	84
$\bar{B}$	63.57	64.01	63.97	64.11	63.9
$P(\%)$	49.66	50	49.97	50.08	49.92
ΔB	5.19	5.61	5.6	5.59	5.71
$\Delta P(\%)$	4.06	4.38	4.37	4.37	4.46

TABLE XI
STATISTICS FOR 160-BIT HASH CODES, TYPE B TEST

	N=256	N=512	N=1024	N=2048	N=50×160
B_{min}	66	62	60	57	57
B_{max}	94	96	101	101	108
$\bar{B}$	80.06	80.64	79.94	80.52	80.29
$P(\%)$	50.03	50.4	49.96	50.32	50.18
ΔB	6.17	6.16	6.35	6.66	6.53
$\Delta P(\%)$	3.85	3.85	3.96	4.16	4.08

TABLE XII
STATISTICS FOR 256-BIT HASH CODES, TYPE B TESTS

	N=256	N=512	N=1024	N=2048	N=50×256
B_{min}	108	107	106	104	100
B_{max}	143	148	151	153	156
$\bar{B}$	128.23	129.23	128.6	128.98	128.26
$P(\%)$	50.09	50.47	50.23	50.38	50.1
ΔB	8.44	8.12	8.05	8.08	8.06
$\Delta P(\%)$	3.29	3.17	3.14	3.15	3.14

TABLE XIII
STATISTICS FOR 512-BIT HASH CODES, TYPE B TESTS

	N=256	N=512	N=1024	N=2048	N=50×512
B_{min}	229	229	229	217	216
B_{max}	278	292	292	300	300
$\bar{B}$	255.28	255.82	256.45	256.62	256.11
$P(\%)$	49.86	49.96	50.08	50.12	50.02
ΔB	11.95	11.42	11.32	11.29	11.31
$\Delta P(\%)$	2.33	2.23	2.21	2.2	2.2

TABLE XIV
STATISTICS FOR 1024-BIT HASH CODES, TYPE B TESTS

	N=256	N=512	N=1024	N=2048	N=50×1024
B_{min}	473	472	464	458	441
B_{max}	551	556	556	557	568
$\bar{B}$	510.56	511.32	512.41	512.32	511.61
$P(\%)$	49.85	49.93	50.04	50.03	49.96
ΔB	13.7	14.82	15.16	15.85	15.95
$\Delta P(\%)$	1.33	1.44	1.48	1.54	1.55

TABLE XV
COMPARISON OF STATISTICS FOR 128-BIT HASH CODES AND N=2048, TYPE B TESTS

	B_{min}	B_{max}	$\bar{B}$	$P(\%)$	ΔB	$\Delta P(\%)$
SHAH	43	83	64.11	50.08	5.59	4.37
Ref. [10]	47	81	63.95	49.96	5.62	4.39
Ref. [11]	48	83	64.22	50.17	5.65	4.42
Ref. [13]	47	84	63.94	49.95	5.69	4.44

TABLE XVI
ABSOLUTE DIFFERENCE D FOR HASH CODES, TYPE A TESTS, WHERE $N = 10,000$.

n	Maximum	Minimum	Mean
128	2386	537	1367
160	2821	717	1706
256	4049	1395	2731
512	7517	3922	5459

TABLE XVII
COMPARISON OF THE ABSOLUTE DIFFERENCE FOR 128-HASH CODES, TYPE A TESTS, WHERE $N = 10,000$.

n	Maximum	Minimum	Mean
SHAH	2386	537	1367
Ref. [10]	2391	656	1364
Ref. [13]	2320	737	1494
Ref. [22]	2455	599	1439
Ref. [33]	2064	655	1367

results demonstrate that the suggested hashing algorithm has very robust potential for confusion and diffusion. Thus, the SHAH function is trustworthy against this kind of attacks.

E. Collision Analysis

In this section we will analyse the novel hash function SHAH based on the collision tests proposed in [10]. In general, a common characteristic of a hash scheme is to have a collision resistance capability, the following two types of tests are performed, type A and type B. In tests of type A, an input stream of size $L = 50n$ is generated and its corresponding $n-$bit hash code is calculated and saved in ASCII format. Then, a new stream is used by choosing an one bit at random from the input message and modified to 0 if it is 1 or to 1 if it is 0. The n-bit hash code of the new stream is calculated and recorded in ASCII format. The two hash codes are compared, and the number of ASCII symbols with the same value at the same location is counted. The absolute difference D between the two hash codes is computed by the following: $D = \sum_{i=1}^{n/8} |dec(e_i) - dec(e_i^{'})|$, where e_i and $e_i^{'}$ be the i-th entry of the input and new hash code, respectively, and function $dec()$ converts the entries to their equivalent decimal values. The test of type A is repeated $N = 10,000$ times, and minimum, maximum, and mean of D are presented in Table XVI for different hash codes of size $n =$ 128, 160, 256, and 512.

Table XVII outlines the absolute differences of 128-hash codes generated under tests of type A, where $N = 10,000$, of some existing hash functions which are based on chaotic maps. The results show that the SHAH has comparable values.

The number of hits where the ASCII symbols are equal, where $N = 10,000$ and the hash codes are generated under tests of type A, is listed in Table XVIII and distribution of the 128-hash codes, is presented in Figure 4.

In the type B tests, an input message M of a fixed size $L = 50n$ bits is created at random and its corresponding n-bit input hash code is computed. Then, a single bit of the

TABLE XVIII
Count of hits in Collision test for $N = 10,000$, type A tests

n	0	1	2	3	4	5
128	9393	595	12	0	0	0
160	9236	734	30	0	0	0
256	8795	1138	64	3	0	0
512	7789	1944	242	23	0	0

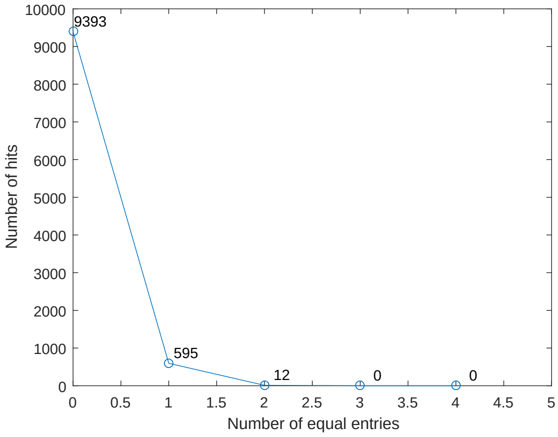


Fig. 4. Distribution of the values of locations where the ASCII symbols are equal in the 128-bit hash codes, type A tests, where $N = 10,000$

input stream is chosen, modified to 0 if it is 1 or to 1 if it is 0, and the hash code of the modified stream is calculated. Table XIX presents minimum, maximum, and mean values of D for different hash codes of size $n = 128, 160, 256,$ and 512 calculated under tests of type B. The same input message is used for all N steps. Comparison with other algorithm is presented in Table XX.

Distribution of the values of locations where the ASCII symbols are equal in the 128-bit hash codes calculated under tests of type B, where $N = 50 \times 128 = 6400$ are presented in Figure 5.

TABLE XIX
Absolute difference D for hash codes, type B tests, where $N = 50n$.

n	Maximum	Minimum	Mean	N
128	2035	636	1248	6400
160	2830	969	1908	8000
256	4124	1483	2817	12800
512	7469	3726	5709	25600

TABLE XX
Comparison of absolute difference D for 128-hash codes, type B tests, where $N = 6400$.

n	Maximum	Minimum	Mean
SHAH	2035	636	1248
Ref. [10]	2421	735	1576
Ref. [11]	2294	661	1360

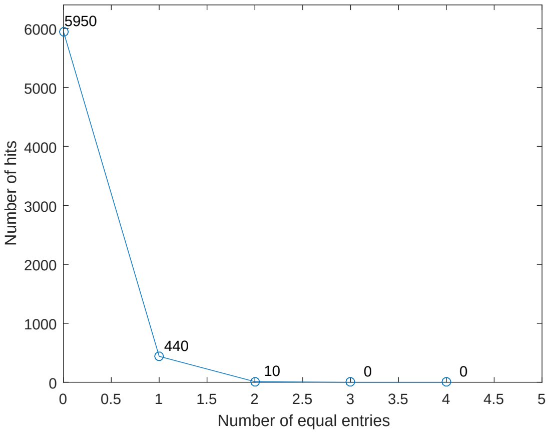


Fig. 5. Distribution of the values of locations where the ASCII symbols are equal in the 128-bit hash codes, type B tests, where $N = 6400$

TABLE XXI
Absolute difference D for hash codes, type B tests, where $N = n$.

n	Maximum	Minimum	Mean	N
128	1812	844	1263	128
160	2480	1253	1840	160
256	3894	2065	2843	256
512	7469	4400	5695	512

In addition to the above operations, the tests of type B are repeated for very small input strings consisting of a single $n - bit$ block, Table XXI.

Distribution of the values of locations where the ASCII symbols are equal in the 128-bit hash codes, type B tests, where $N = 128$ and $L = n$ are presented in Figure 6.

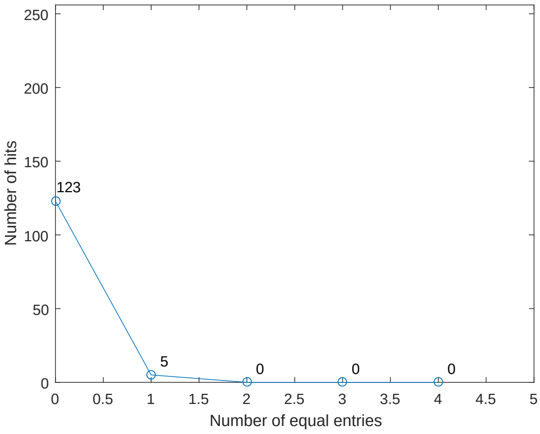


Fig. 6. Distribution of the values of locations where the ASCII symbols are equal in the 128-bit hash codes, type B tests, where $N = 128$ and $L = n$

From the obtained results it is clear that the novel hash algorithm SHAH has a strong collision resistance capacity. Compared with similar hash functions, the proposed one has a mean per character values close to the ideal of 85.3333 [7] and low collision values.

IV. CONCLUSIONS

In this article, we present a novel hash algorithm based on shrinking chaotic function. The hash function called SHAH is based on two Tinkerbell maps filtered with the decimation rule. Exact research has been provided on the novel scheme using distribution analysis, sensitivity analysis, static analysis of diffusion, static analysis of confusion, and collision analysis. The experimental data show very good statistical results and high level of cryptographic security of the SHAH algorithm.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their valuable comments.

REFERENCES

[1] M. Ahmad, S. Khurana, S. Singh, and H.D. AlSharari, "A simple secure hash function scheme using multiple chaotic maps", *3D Research* 8 (13), 2129–2151 (2017).

[2] K.T. Alligood, T.D. Sauer, J.A. Yorke, "CHAOS: An introduction to dynamical systems", Springer-Verlag, Berlin, 1996.

[3] G. Alvarez and S. Li, "Some basic cryptographic requirements for chaos-based cryptosystems", *International Journal of Bifurcation and Chaos* 16, 2129–2151 (2006).

[4] P. Apostoli and A. Kanda, "Cantorian sets, fuzzy sets, rough sets and fregean sets", *Bull. Pol. Ac.: Tech.* 50 (3), 247–276 (2002).

[5] L. Bassham, A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, N. Heckert, and J. Dray, "A Statistical test suite for random and pseudorandom number generators for cryptographic application", *NIST Special Publication 800-22*, Revision 1a (Revised: April 2010), http://doi.org/10.6028/NIST.SP.800-22r1a.

[6] D. Coppersmith, H. Krawczyk, Y., Mansour, "The shrinking generator", in Advances in Cryptology: CRYPTO 93, Springer Verlag, Lecture Notes in Computer Science, vol. 773, 1994, pp. 22-39.

[7] S. Deng, Y. Li, and D. Xiao, "Analysis and improvement of a chaos-based Hash function construction", *Communications in Nonlinear Science and Numerical Simulation* 15, 1338–1347 (2010).

[8] M. Francois, T. Grosges, D. Barchiesi, and R. Erra, "A new pseudo-random number generator based on two chaotic maps", *Informatica* 24 (2), 181–197 (2013).

[9] H. Hristov, "Scanning for vulnerabilities in the security mechanisms of the hosts in the academic institutions and government agencies", *Mathematical and Software Engineering* 4 (1), 1-6 (2018).

[10] A. Kanso and M. Ghebleh, "A fast and efficient chaos-based keyed hash function", *Communications in Nonlinear Science and Numerical Simulation* 18 (1), 109–123 (2013).

[11] A. Kanso and M. Ghebleh, "A structure-based chaotic hashing scheme", *Nonlinear Dynamics* 81 (1-2), 27–40 (2015).

[12] A. Kanso and N. Smaoui, "Irregularly decimated chaotic map(s) for binary digit generation", *International Journal of Bifurcation and Chaos* 19 (4), 1169–1183 (2009).

[13] A Kanso, H. Yahyaoui, and M. Almulla, "Keyed hash function based on chaotic map", *Information Sciences* 186, 249–264 (2012).

[14] K. Kordov, "Modified pseudo-random bit generation scheme based on two circle maps and XOR function", *Applied Mathematical Sciences* 9 (3), 129–135 (2015).

[15] K. Kordov and L. Bonchev, "Using circle map for audio encryption Algorithm", *Mathematical and Software Engineering* 3 (2), 183–189 (2017).

[16] H.S. Kwok and W.S.T. Tang, "A chaos-based cryptographic hash function for message authentication", *International Journal of Bifurcation and Chaos* 15 (12), 4043–4050 (2005).

[17] D. Lambić and M. Nikolić, "Pseudo-random number generator based on discrete-space chaotic map", *Nonlinear Dynamics* 90 (1), 223–232 (2017).

[18] Y. Li, Di. Xiao, and S. Deng, "Hash function based on the chaotic look-up table with changeable parameter", *International Journal of Modern Physics B* 25 (29), 3835–3851 (2011).

[19] 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", *International Journal of Bifurcation and Chaos* 27 (7), 1750106 (2017).

[20] G. Marsaglia, *DIEHARD: a battery of tests of randomness*, 1995, https://github.com/reubenhwk/diehard.

[21] W. Meier and O. Staffelbach, "The self-shrinking generator", In: Blahut R.E., Costello D.J., Maurer U., Mittelholzer T. (eds) Communications and Cryptography, Springer, Boston, MA, The Springer International Series in Engineering and Computer Science (Communications and Information Theory), vol. 276, 1994, pp. 287–295.

[22] H. Ren, Y. Wang, Q. Xie, and H. Yang, "A novel method for one-way hash function construction based on spatiotemporal chaos", *Chaos, Solitons and Fractals* 42, 2014–2022 (2009).

[23] C.E. Shannon, "Communication theory of secrecy systems", *Bell System Technical Journal* 27 (4), 656–715 (1949).

[24] B.P. Stoyanov, "Chaotic cryptographic scheme and its randomness evaluation", *in 4th AMiTaNS'12*, AIP Conference Proceedings, 1487, 2012, pp. 397–404, doi: 10.1063/1.4758983.

[25] B.P. Stoyanov, "Pseudo-random bit generator based on Chebyshev map", in 5th AMiTaNS'13, AIP Conference Proceedings, vol. 1561, 2013, pp. 369–372.

[26] B.P. Stoyanov, "Pseudo-random bit generation algorithm based on Chebyshev polynomial and Tinkerbell map", *Applied Mathematical Sciences* 8 (125), 6205–6210 (2014).

[27] B.P. Stoyanov, "Using circle map in pseudorandom bit generation", in 6th AMiTaNS'14, AIP Conference Proceedings, vol. 1629, 2014, pp. 460–463, doi: 10.1063/1.4902309.

[28] B. Stoyanov, K. Szczypiorski, and K. Kordov, "Yet another pseudo-random number generator", *International Journal of Electronics and Telecommunications* 63 (2), 195–199 (2017).

[29] B. Stoyanov and K. Kordov, "A novel pseudorandom bit generator based on Chirikov standard map filtered with shrinking rule", *Mathematical Problems in Engineering* 2014, Article ID 986174 (2014), http://dx.doi.org/10.1155/2014/986174.

[30] B. Stoyanov, and K. Kordov, "Novel secure pseudo-random number generation scheme based on two Tinkerbell maps", *Advanced Studies in Theoretical Physics* 9 (9), 411–421, 2015, http://dx.doi.org/10.12988/astp.2015.5342.

[31] Li Yantao and Li Xiang, "Chaotic hash function based on circular shifts with variable parameters", *Chaos, Solitons and Fractals* 91, 639–648 (2016).

[32] J. Walker, *ENT: a pseudorandom number sequence test program*, http://www.fourmilab.ch/random/.

[33] Y. Wang, K-W. Wong, and Di Xiao, "Parallel hash function construction based on coupled map lattices", *Communications in Nonlinear Science and Numerical Simulation* 16, 2810–2821 (2011).

[34] IEEE Computer Society (2008), 754-2008 - IEEE standard for floating-point arithmetic, *Revision of ANSI/IEEE Std 754-1985*, 10.1109/IEEESTD.2008.4610935.