www.czasopisma.pan.pl www.journals.pan.pl

BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 67, No. 3, 2019
DOI: 10.24425/bpasts.2019.129654

BentSign: keyed hash algorithm based on bent Boolean function and chaotic attractor

M. TODOROVA[1], B. STOYANOV[1], K. SZCZYPIORSKI[2], W. GRANISZEWSKI[2], and K. KORDOV[1]

[1] Konstantin Preslavsky University of Shumen, 115 Universitetska Str., 9712 Shumen, Bulgaria
[2] Warsaw University of Technology, Pl. Politechniki 1, 00-661 Warsaw, Poland

Abstract. In this study, we propose a novel keyed hash algorithm based on a Boolean function and chaotic attractor. The hash algorithm called BentSign is based on two Signature attractors and XOR function and a bent Boolean function. The provided theoretical and experimental results confirm that the novel scheme can generate output hashes with a good level of security, collision resistance, and protection against most common attacks.

Key words: hash algorithm, chaotic attractor, Bent Boolean function, pseudorandom bit generation scheme.

1. Introduction

Keyed hash algorithms play an important role in communication services. They map data of arbitrary lengths to data of fixed length. Applications include message authentication, password storage, and data integrity. The cryptographic hash algorithms were introduced in [8] by Diffie and Hellman for using in public key infrastructure. They are characterized by: (a) sensitivity, (b) collision resistance, (c) uniform distribution, and (d) static confusion and diffusion properties.

In [6, 14, 17, 20, 21, 28–31], and [36], a number of generation schemes and program tools based on chaotic attractors and shift registers, are proposed. They can be successfully integrated in social networks [19] and [37].

In recent years, high dimensional chaotic systems have been admirable scientific area and have got much emphasis in data transmissions field. In [9], an image encryption constructed by eight dimensional chaotic map is proposed. In [39], a video encryption algorithm based on the twelve-dimensional chaotic map and the Ikeda delay differential equation is presented. The authors of [1] used high dimensional map to present novel parallel hash algorithm. A novel hash function based on a four-dimensional hyperchaotic Lorenz system is designed in [22]. Other examples with research about high dimensional chaos based hash algorithms can be found in [10–12, 15], and [38]. The high dimensional chaotic systems have significant characteristics such as difficult dynamic reconstruction, long period without any doubling, and ergodicity [1, 4].

In our humble opinion, the main contributions of our work can be summarized as follows:

- We propose novel pseudorandom bit algorithm based on two Signature attractors, which has good statistical options;
- We combine the pseudorandom generator with bent Boolean function to novel keyed hash algorithm;
- We evaluate the proposed algorithm, and the results illustrate that it has good capabilities of confusion and diffusion, strong collision resistance and has desirable security properties that can withstand most common attack such as birthday attack, meet-in-the-middle attack, and pre-image attack.

In Section 2 we present a novel pseudorandom output bit scheme based on two Signature attractors. In Section 3 we provide some introduction to bent Boolean functions. In Section 4 we present the novel hash algorithm BentSign and complete security analysis is given. Finally, the last section concludes the paper.

2. Pseudorandom bit construction based on chaotic attractor

The work presented in this section was motivated by recent developments in chaos-based pseudorandom generation [13,32–34], and [35].

2.1. Proposed pseudorandom bit generation scheme. The Signature attractor is presented in [26], Eq. (1):

$$
\begin{aligned}
x_{t+1} &= x_t \cos\theta_t - y_t \sin\theta_t + 1 - 0.8 x_t z_t\,,\\
y_{t+1} &= x_t \sin\theta_t + y_t \cos\theta_t\,,\\
z_{t+2} &= 1.4 z_{t+1} + 0.3 z_t (1 - z_t),\\
\theta_t &= 5.5 - \frac{1}{\sqrt{x_t^2 + y_t^2 + z_t^2}}.
\end{aligned}
\tag{1}
$$

*e-mail: ksz@tele.pw.edu.pl

Manuscript submitted 2018-08-25, revised 2018-11-27, initially accepted for publication 2018-12-14, published in June 2019.

With initial values (x_i, y_i), for $i = 150..950$, $x_0 = -0.7390212$, $y_0 = -0.7244441$, $z_0 = 0.3999123$, and $z_1 = 1.454601$, two-dimensional model of Signature attractor is illustrated in Fig. 1, and three-dimensional models viewed from two different angles are plotted in Fig. 2.

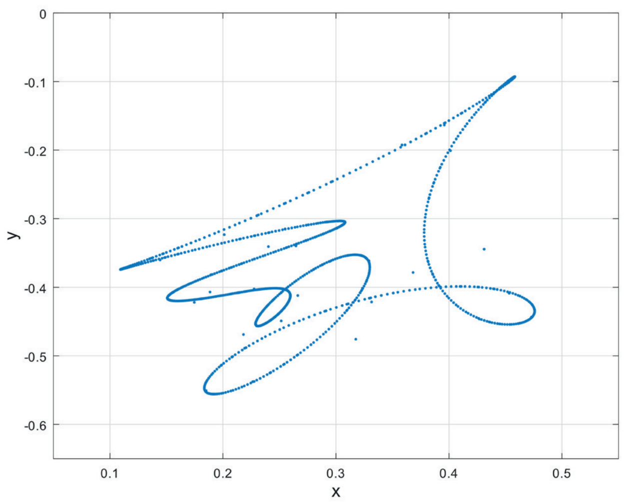

Fig. 1. The two-dimensional plot of Signature attractor of Eq. (1), where $t = 150..950$; $x - y$ plane

The new generator is slightly modified scheme of the algorithm presented in [13]. It is based on the following two Signature attractors:

$$
\begin{aligned}
x_{0,t+1} &= x_{0,t}\cos\theta_{0,t} - y_{0,t}\sin\theta_{0,t} + 1 - 0.8x_{0,t}z_{0,t}\,,\\
y_{0,t+1} &= x_{0,t}\sin\theta_{0,t} + y_{0,t}\cos\theta_{0,t}\,,\\
z_{0,t+2} &= 1.4z_{0,t+1} + 0.3z_{0,t}(1 - z_{0,t})\\
\theta_{0,t} &= 5.5 - \frac{1}{\sqrt{x_{0,t}^2 + y_{0,t}^2 + z_{0,t}^2}}\,,\\
x_{1,t+1} &= x_{1,t}\cos\theta_{1,t} - y_{1,t}\sin\theta_{1,t} + 1 - 0.8x_{1,t}z_{1,t}\,,\\
y_{1,t+1} &= x_{1,t}\sin\theta_{1,t} + y_{1,t}\cos\theta_{1,t}\,,\\
z_{1,t+2} &= 1.4z_{1,t+1} + 0.3z_{1,t}(1 - z_{1,t}),\\
\theta_{1,t} &= 5.5 - \frac{1}{\sqrt{x_{1,t}^2 + y_{1,t}^2 + z_{1,t}^2}}
\end{aligned}
\tag{2}
$$

and is based on the following steps:

1. The elements $x_{0,0}, y_{0,0}, z_{0,0}, z_{0,1}, x_{1,0}, y_{1,0}, z_{1,0}$, and $z_{1,1}$ from Eq. (2) are initialized with values.
2. The attractors from Eq. (2) are iterated for L_0 and L_1 times.
3. The iteration of the Eq. (2) progresses, and we get six real fractions $x_{0,m}$, $y_{0,m}$, $z_{0,m+1}$, $x_{1,n}$, $y_{1,n}$, and $z_{1,n+1}$, which are post-processed as follows:

$$
\begin{aligned}
s_1 &= mod(abs(integer(x_{0,m} \times 10^7)), 2),\\
s_2 &= mod(abs(integer(y_{0,m} \times 10^7)), 2),\\
s_3 &= mod(abs(integer(z_{0,m+1} \times 10^7)), 2),\\
s_4 &= mod(abs(integer(x_{1,n} \times 10^7)), 2),
\end{aligned}
$$

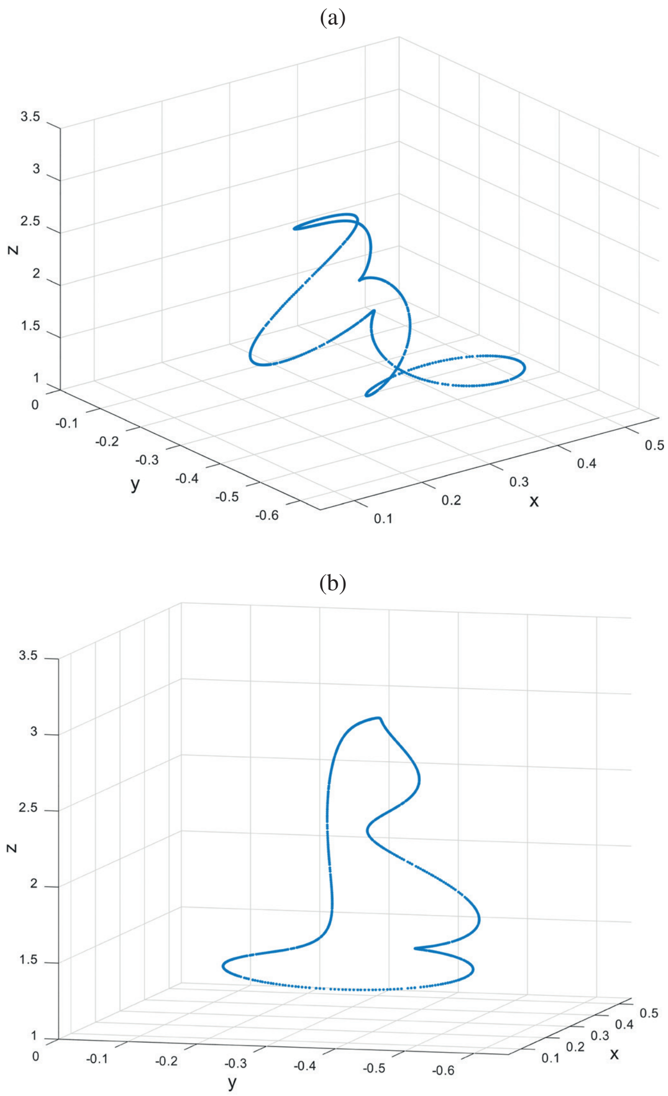

Fig. 2. The three-dimensional plots of Signature attractor of Eq. (1), where $t = 150..950$

$$
\begin{aligned}
s_5 &= mod(abs(integer(y_{1,n} \times 10^7)), 2),\\
s_6 &= mod(abs(integer(z_{1,n+1} \times 10^7)), 2),
\end{aligned}
$$

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

4. Perform logical XOR between s_1, s_2, s_3, s_4, s_5, and s_6 to get a single output bit.
5. Return to Step 3 until the bit stream limit is reached.

The proposed pseudorandom bit generation scheme is implemented in C++ programming language, using the following initial values: $x_{0,0} = -0.4584282$, $y_{0,0} = -1.7876741$, $z_{0,0} = 0.1964$, $z_{0,1} = 1.020591$, $x_{1,0} = -0.7390212$, $y_{1,0} = -2.7244441$, $z_{1,0} = 0.3999123$, $z_{1,1} = 1.454601$, $L_0 = 714$, and $L_1 = 1278$.

2.2. Key space analysis. The set of all initial values compose the key space. The novel generation scheme has eight input parameters $x_{1,0}$, $y_{1,0}$, $x_{2,0}$, and $y_{2,0}$. According to [42], the computational precision of the 64-bit double-precision number is about 10^{-15}, thus the key set is $(10^{15})^8$ bits, which is more than 2^{398} bits. The novel pseudorandom generator posses a large key space to defend against exhaustive key search attack [2]. Moreover, the initial iteration values L_0 and L_1 can also be used as a part of the key space.

2.3. Statistical tests. In order to measure randomness of the sequences of bits produced by the new pseudorandom number algorithm, we used the statistical applications NIST [5], DIEHARD [16], and ENT [40].

For the NIST suite, we outputted 1000 binary series of length $1,000,000$ bits. The results from the tests are given in Table 1. The minimum pass rate for each statistical test with the exception of the Random excursion (variant) test is approximately 980 for a sample size of 1000 binary sequences. The minimum pass rate for the Random excursion (variant) test is approximately 618 for a sample size of 632 binary sequences. The output bits from the proposed pseudorandom bit generator scheme passed successfully all the NIST tests.

Table 1
NIST test suite results

NIST test	P-value	Pass rate
Frequency (monobit)	0.190654	989/1000
Block-frequency	0.382115	991/1000
Cumulative sums (Forward)	0.095426	988/1000
Cumulative sums (Reverse)	0.278461	988/1000
Runs	0.803720	991/1000
Longest run of Ones	0.755819	981/1000
Rank	0.371941	995/1000
FFT	0.498313	989/1000
Non-overlapping templates	0.479685	990/1000
Overlapping templates	0.057510	982/1000
Universal	0.378705	991/1000
Approximate entropy	0.542228	986/1000
Random-excursions	0.347462	625/632
Random-excursions Variant	0.658650	625/632
Serial 1	0.270265	995/1000
Serial 2	0.339271	991/1000
Linear complexity	0.100109	989/1000

The DIEHARD application is a set of 19 statistical tests and they return $P-values$, which should be uniform in $[0,1)$, if the input stream contains pseudorandom 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. BentSign algorithm passed successfully all DIEHARD tests, Table 2.

The ENT software includes following tests: entropy, optimum compression, χ^2 distribution, arithmetic mean value, Monte Carlo value for π, and serial correlation coefficient. Sequences of bytes are stored in files. We tested output series of 125,000,000 bytes of the BentSign. The algorithm passed successfully all ENT test, Table 4, and Table 5, and Table 3.

Table 2
DIEHARD statistical test results

DIEHARD test	P-value
Birthday spacings	0.411472
Overlapping 5-permutation	0.619766
Binary rank (31×31)	0.887539
Binary rank (32×32)	0.371790
Binary rank (6×8)	0.493264
Bitstream	0.625534
OPSO	0.439096
OQSO	0.386868
DNA	0.513739
Stream count-the-ones	0.341532
Byte count-the-ones	0.484595
Parking lot	0.586842
Minimum distance	0.444122
3D spheres	0.502462
Squeeze	0.970210
Overlapping sums	0.585222
Runs up	0.519402
Runs down	0.409812
Craps	0.634792

Table 3
ENT statistical test results

ENT test	Results
Entropy	7.999999 bits per byte
Optimum compression	OC would reduce the size of this 125000000 byte file by 0%
χ^2 distribution	For 125000000 samples is 233.87, and randomly would exceed this value 82.47% of the time
Arithmetic mean value	127.4979 (127.5 = random)
Monte Carlo π estim.	3.141588914 (error 0.00%)
Serial correl. coeff.	0.000111 (totally uncorrelated = 0.0)

Based on the good test results, we can conclude that the novel pseudorandom bit generation algorithm has satisfying statistical properties and provides acceptable level of security.

Table 4
Number of occurrences of byte values from 0 to 127, and the fraction of the overall ENT test file made up by that value

Value	Occurrences	Fraction	Value	Occurrences	Fraction	Value	Occurrences	Fraction	Value	Occurrences	Fraction
0	488434	0.003907	32	488970	0.003912	64	488601	0.003909	96	487327	0.003899
1	488041	0.003904	33	488365	0.003907	65	488914	0.003911	97	487886	0.003903
2	487859	0.003903	34	487089	0.003897	66	489029	0.003912	98	488817	0.003911
3	488381	0.003907	35	487947	0.003904	67	488325	0.003907	99	488313	0.003907
4	487539	0.0039	36	489023	0.003912	68	488872	0.003911	100	489391	0.003915
5	488168	0.003905	37	489400	0.003915	69	487761	0.003902	101	487559	0.0039
6	488817	0.003911	38	487621	0.003901	70	488248	0.003906	102	488578	0.003909
7	489126	0.003913	39	488079	0.003905	71	489001	0.003912	103	488247	0.003906
8	489424	0.003915	40	487768	0.003902	72	488352	0.003907	104	487674	0.003901
9	488326	0.003907	41	487504	0.0039	73	488962	0.003912	105	488875	0.003911
10	487377	0.003899	42	490046	0.00392	74	488285	0.003906	106	488122	0.003905
11	487978	0.003904	43	488676	0.003909	75	487644	0.003901	107	486532	0.003892
12	489190	0.003914	44	487995	0.003904	76	489441	0.003916	108	487817	0.003903
13	488502	0.003908	45	487208	0.003898	77	487622	0.003901	109	487480	0.0039
14	487949	0.003904	46	488932	0.003911	78	488530	0.003908	110	489642	0.003917
15	488637	0.003909	47	488470	0.003908	79	488261	0.003906	111	487480	0.0039
16	488322	0.003907	48	488692	0.00391	80	488534	0.003908	112	488491	0.003908
17	488490	0.003908	49	488422	0.003907	81	487814	0.003903	113	487942	0.003904
18	488391	0.003907	50	488813	0.003911	82	488796	0.00391	114	487759	0.003902
19	488586	0.003909	51	487790	0.003902	83	487018	0.003896	115	488511	0.003908
20	487332	0.003899	52	487904	0.003903	84	489481	0.003916	116	488991	0.003912
21	488497	0.003908	53	489553	0.003916	85	487332	0.003899	117	488032	0.003904
22	488036	0.003904	54	487598	0.003901	86	487919	0.003903	118	489352	0.003915
23	488123	0.003905	55	488752	0.00391	87	488090	0.003905	119	488394	0.003907
24	488078	0.003905	56	489107	0.003913	88	488140	0.003905	120	488113	0.003905
25	487889	0.003903	57	488204	0.003906	89	488045	0.003904	121	487815	0.003903
26	488790	0.00391	58	488280	0.003906	90	486882	0.003895	122	488694	0.00391
27	489068	0.003913	59	488376	0.003907	91	488770	0.00391	123	488863	0.003911
28	487177	0.003897	60	487683	0.003901	92	487961	0.003904	124	488895	0.003911
29	487529	0.0039	61	489153	0.003913	93	488357	0.003907	125	487743	0.003902
30	489931	0.003919	62	488148	0.003905	94	489199	0.003914	126	487995	0.003904
31	487215	0.003898	63	487322	0.003899	95	488848	0.003911	127	488685	0.003909

Table 5
Number of occurrences of byte values from 128 to 255, and the fraction of the overall ENT test file made up by that value

Value	Occurrences	Fraction	Value	Occurrences	Fraction	Value	Occurrences	Fraction	Value	Occurrences	Fraction
128	488965	0.003912	160	488106	0.003905	192	488417	0.003907	224	487036	0.003896
129	487572	0.003901	161	487668	0.003901	193	488296	0.003906	225	488236	0.003906
130	488400	0.003907	162	487549	0.0039	194	488275	0.003906	226	489551	0.003916
131	488324	0.003907	163	489289	0.003914	195	488354	0.003907	227	488558	0.003908
132	487821	0.003903	164	487296	0.003898	196	488052	0.003904	228	489241	0.003914
133	488265	0.003906	165	488344	0.003907	197	489443	0.003916	229	487794	0.003902
134	487534	0.0039	166	488156	0.003905	198	488413	0.003907	230	488209	0.003906
135	488290	0.003906	167	488168	0.003905	199	487965	0.003904	231	487687	0.003901
136	486413	0.003891	168	488912	0.003911	200	488261	0.003906	232	488684	0.003909
137	486966	0.003896	169	488210	0.003906	201	490119	0.003921	233	489030	0.003912
138	488850	0.003911	170	488477	0.003908	202	487254	0.003898	234	488585	0.003909
139	488185	0.003905	171	487564	0.003901	203	489182	0.003913	235	488461	0.003908
140	487509	0.0039	172	489899	0.003919	204	487792	0.003902	236	487620	0.003901
141	489092	0.003913	173	488677	0.003909	205	489208	0.003914	237	488086	0.003905
142	487988	0.003904	174	489141	0.003913	206	488935	0.003911	238	487762	0.003902
143	486979	0.003896	175	487903	0.003903	207	488006	0.003904	239	487811	0.003902
144	488299	0.003906	176	487777	0.003902	208	486490	0.003892	240	488139	0.003905
145	488921	0.003911	177	488750	0.00391	209	487821	0.003903	241	488213	0.003906
146	488418	0.003907	178	488383	0.003907	210	487826	0.003903	242	487737	0.003902
147	487972	0.003904	179	487878	0.003903	211	489176	0.003913	243	488624	0.003909
148	487476	0.0039	180	487760	0.003902	212	489027	0.003912	244	488013	0.003904
149	488524	0.003908	181	489304	0.003914	213	487546	0.0039	245	487734	0.003902
150	488133	0.003905	182	488200	0.003906	214	488207	0.003906	246	488646	0.003909
151	488231	0.003906	183	488435	0.003907	215	488672	0.003909	247	488245	0.003906
152	488906	0.003911	184	488561	0.003908	216	488826	0.003911	248	487925	0.003903
153	488220	0.003906	185	487562	0.0039	217	488481	0.003908	249	488648	0.003909
154	488151	0.003905	186	487178	0.003897	218	487678	0.003901	250	488224	0.003906
155	487810	0.003902	187	489543	0.003916	219	488522	0.003908	251	487302	0.003898
156	487332	0.003899	188	487620	0.003901	220	488148	0.003905	252	488137	0.003905
157	489194	0.003914	189	487508	0.0039	221	489439	0.003916	253	488672	0.003909
158	489802	0.003918	190	487486	0.0039	222	488162	0.003905	254	487645	0.003901
159	488559	0.003908	191	488429	0.003907	223	488405	0.003907	255	489372	0.003915

3. Bent Boolean functions

In this section we refer to works of [7, 23], and [25].

Definition 1. A Boolean function f in n variables is map from $\mathbb{V}_n$ (the vector space on n dimension) to the two-element Galois field $\mathbb{F}_2$. The (0,1)-sequence defined by $(f(\mathbf{x}_0), f(\mathbf{x}_1), \ldots, f(\mathbf{x}_{2^n-1}))$ is called the truth table of f, where $\mathbf{v}_0 = (0,\ldots,0,0)$, $\mathbf{v}_1 = (0,\ldots,0,1)$, $\ldots$, $\mathbf{v}_{2^n-1} = (1,\ldots,1,1)$, lexicographically sorted list.

To each Boolean function $f : \mathbb{V}_n \to \mathbb{F}_n$ we associate *sign* function, denoted by $\hat{f} : \mathbb{V}_n \to \mathbb{R}^* \subseteq \mathbb{C}^*$ and defined by $\hat{f}(\mathbf{x}) = (-1)^{f(\mathbf{x})}$.

Definition 2. The Walsh transform of a function f on $\mathbb{V}_n$ (with the values of f taken to be real numbers 0 and 1) is the map $W(f) : \mathbb{V}_n \to \mathbb{R}$, defined by

$$W(f)(\mathbf{w}) = \sum_{\mathbf{x}\in\mathbb{V}_n} f(\mathbf{x})(-1)^{\mathbf{w}\cdot\mathbf{x}},$$

which defines the coefficients of f with respect to the orthonormal basis of the group characters $Q_{\mathbf{x}}(\mathbf{w}) = (-1)^{\mathbf{w}\cdot\mathbf{x}}$ (where $\mathbf{w}\cdot\mathbf{x}$ is the scalar product); f can be recovered by the inverse Walsh transform

$$f(\mathbf{x}) = 2^{-n} \sum_{\mathbf{w}\in\mathbb{V}_n} W(f)(\mathbf{w})(-1)^{\mathbf{w}\cdot\mathbf{x}}.$$

Definition 3. A Boolean function f in n variables is called *bent* if and only if the Walsh transform coefficients of $\hat{f}$ are all $\pm 2^{n/2}$, that is, $W(\hat{f})^2$ is constant.

Lemma 1. If a bent function f exists, then n must be even, $n = 2k$.

They are designed two classes of bent Boolean functions on $\mathbb{V}_n$, $n = 2k$. The first one is: Let $x_1, y_1, \ldots, x_k, y_k$ be independent variables and let $P(\mathbf{x})$ be an arbitrary polynomial, where $\mathbf{x} = (x_1, \ldots, x_k)$. Then

$$f_1(x_1, y_1, \ldots, x_k, y_k) = \sum_{i=1}^{k} x_i y_i \oplus P(\mathbf{x})$$

is bent function. This generalizes Corollary 2.

Lemma 2. The function

$$f(\mathbf{x}) = x_1 x_2 \oplus \cdots \oplus x_{2k-1} x_{2k},\ k \geq 1, \tag{3}$$

is bent.

4. Hash algorithm based on bent Boolean function and chaotic attractor

In this section, we extend the work of Ref. [38] by combining chaotic attractors with bent Boolean function.

4.1. Proposed Keyed hash algorithm based on bent Boolean function and chaotic attractor. In this subsection, the combination of bent Boolean and Signature attractors is used to design a keyed hash algorithm named BentSign.

Let n be the bit of the final hash value. The parameter n usually supports five bit lengths, 128, 160, 256, 512, and 1024 bits. We consider input message (string) M' with arbitrary length. The proposed hash algorithm BentSign consists of the following steps:

1. The input message M is converted into binary sequence using ASCII table.
2. The binary sequence from Step 1 is padded with a single bit of one followed by bits of zeroes until the M' has length m, obtained as multiple of n.
3. The novel pseudorandom bit generation algorithm (Section 2) based on two Signature attractors and XOR function is repeatedly iterated, getting m bits. m-sized vector P is produced.
4. The m-sized vectors M' and P are united in a new m-sized vector, N, using XOR operation.
5. The vector N is split into p blocks, $N_1, N_2, \ldots, N_p$, each of length n and $m = np$ is the total length of the vector N.
6. A temporary n-sized vector T is obtained by $T = N_1 \oplus N_2 \oplus \cdots \oplus N_p$.
7. A temporary n-sized vector U is taken and all of its elements are initialized to 0s.
8. 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 pseudorandom generator based on two Signature attractors and XOR function (Section 2).
9. The novel pseudorandom bit generation algorithm (Section 2) based on two Signature attractors and XOR function is repeatedly iterated, getting n bits. n-sized vector R is produced.
10. Take the Eq. (3) for $k = 2n$, substitute the odd elements with the bits from vector T and the even elements with the bits from vector R, and calculate the function $f(\mathbf{x})$ to take one single bit u.
11. If the current bit u is 1, the vector U is XOR-ed with the next n bits from the novel pseudorandom generator based on two Signature attractors and XOR function (Section 2). If the current bit u is 0, the matrix U is bitwise rotated left by one bit position.
12. Return to Step 8, until the end of the vector T.
13. The final hash value is obtained by $H = T \oplus U$.

The proposed BentSign hash algorithm is implemented in C++ programming language.

4.2. Distribution analysis. In this subsection, we demonstrate uniformly distributed hash values in the compressed range, when $n = 128$. The input message is taken from Ref. [38]:

> 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.

The ASCII code distribution of input message and the corresponding BentSign hexadecimal values are shown in Fig. 3a and 3b. Another input message with the same length but all of blank spaces, is created. The ASCII code distribution of the blank-spaced input message and the corresponding BentSign hexadecimal hash value are shown in Fig. 3c and 3d. The BentSign hash plots, 3b and 3d, are uniformly distributed in compress range even under particularly conditions.

4.3. Sensitivity analysis. To illustrate the sensitivity of BentSign, simulation experiments have been accomplished under the following 10 conditions [38]:

Condition 0: Blank message.

Condition 1: The input string is the same as the one in Section 4.2;

Condition 2: Change the letter *'K'* in the input into *'k'*.

Condition 3: Change the zero in the input to one.

Condition 4: Change the word *'School'* in the input message to *'school'*.

Condition 5: Change the comma ',' in the input string to '.'.

Condition 6: Add a blank space at the end of the input string.

Condition 7: Change the word *'recognized'* in the input string to *'recognize'*.

Condition 8: Subtracts 1×10^{-15} from the input value $x_{1,0}$.

Condition 9: Adds 1×10^{-15} to the input key value $y_{2,0}$.

The corresponding 128-bit hash values in hexadecimal number system are the following:

Condition 0: FAA8C89DD3970E19A3856027FFF3ED79
Condition 1: B26D06E24E790A34A26C1C0E07B55313
Condition 2: 79F521281FDB6DBA897CF3D4A12CB701
Condition 3: 30DDCB05DD103FAB7241D2029CAF9A23
Condition 4: 953A4A42777F2CC24301A0CD6A612AA6
Condition 5: A6AB45F273F67861927CADF07041CDF7
Condition 6: D5F9F9836AAFE5F7018EBAF2DA044594
Condition 7: 0758DC3F6022E3881D4EA3A6004E0061
Condition 8: 0FC1618D766D4F7E69A3807A440DE4EE
Condition 9: 3E8BD43EBF8C31EA6FC819D851DCAAC1

The corresponding binary representation of the hash values are illustrated in Figure 4. The values from sensitivity analysis illustrate that the BentSign possesses high sensitivity to any changes in input messages, which make serious differences of output hashes.

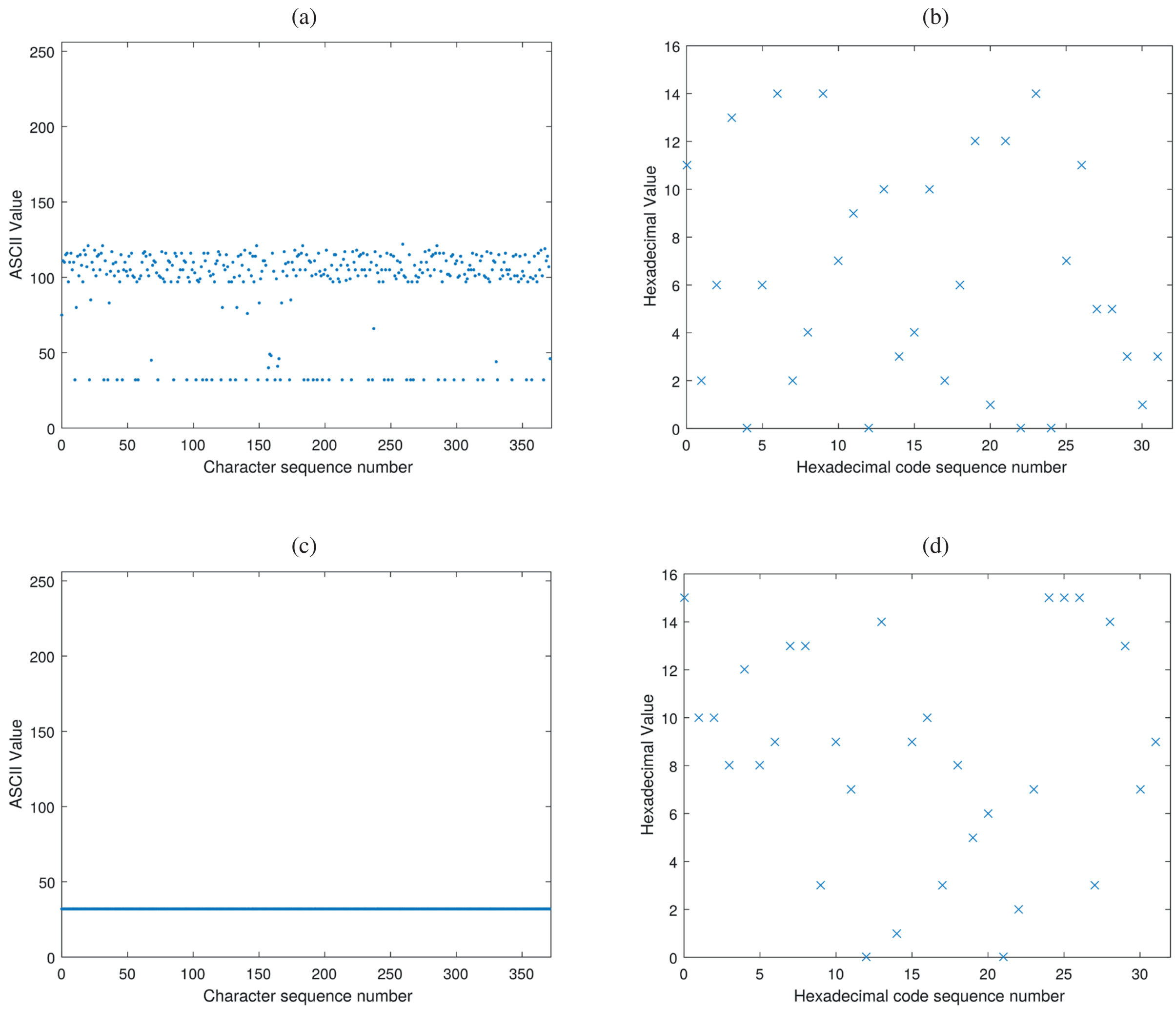

Fig. 3. Distribution of input message and corresponding hash value

Fig. 4. 128-bit hash values of the input messages under ten different conditions: a) Condition 0, b) Condition 1, c) Condition 2, d) Condition 3, e) Condition 4, f) Condition 5, g) Condition 6, h) Condition 7, i) Condition 8, and j) Condition 9

4.4. Statistic analysis of diffusion and confusion Confusion can hide the relationship between the input message and the hash value, and diffusion effect should be that any small changes in the initial data lead to a 50% changing probability of each bit of hash value.

Six functions used here are defined as follows:

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

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

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

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

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

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, 11]: type A and type B. In the type A test, an input string, of size $L = 50n$ is generated and its corresponding $n-$bit hash value is computed. Then, a new string is generated by choosing a single bit at random from the original string and modified to zero if it is one or to one if it is zero. The n-bit hash value of the new string is then compared with that of the original string and the Hamming distance between the two hash values is recorded as B_i. This is then repeated N times, where each time, a new original string is chosen and one of its bits is randomly chosen and modified to zero if it is one or to one if it is zero. Tables 6–10 present results of these tests for n = 128, 160, 256, 512, 1024.

Table 6
Statistical results for 128-bit hash values generated under tests of type A

	N=256	N=512	N=1024	N=2048	N=10,000
$B_{\min}$	49	48	48	48	44
$B_{\max}$	85	85	85	85	85
$\bar{B}$	64.32	63.17	64.07	63.95	63.99
P (%)	50.25	50.13	50.06	49.96	49.99
ΔB	5.68	5.79	5.7	5.68	5.61
ΔP (%)	4.44	4.52	4.45	4.43	4.38

Table 7
Statistical results for 160-bit hash values generated under tests of type A

	N=256	N=512	N=1024	N=2048	N=10,000
$B_{\min}$	59	59	59	59	55
$B_{\max}$	97	97	97	98	102
$\bar{B}$	79.71	80.19	79.74	79.78	79.96
P (%)	49.82	50.12	49.83	49.86	49.98
ΔB	6.6	6.34	6.36	6.22	6.29
ΔP (%)	4.12	3.96	3.97	3.88	3.93

Table 11 lists values from few chaos based hash algorithm. The BentSign has comparable results.

Table 8
Statistical results for 256-bit hash values generated under tests of type A

	N=256	N=512	N=1024	N=2048	N=10,000
$B_{\min}$	104	96	96	96	96
$B_{\max}$	148	153	153	153	158
$\bar{B}$	128.18	127.99	127.64	128.042	127.94
P (%)	50.07	49.99	49.86	50.01	49.97
ΔB	7.78	8.1	7.88	7.97	7.96
ΔP (%)	3.04	3.16	3.08	3.11	3.1

Table 9
Statistical results for 512-bit hash values generated under tests of type A

	N=256	N=512	N=1024	N=2048	N=10,000
$B_{\min}$	215	215	215	215	213
$B_{\max}$	290	290	302	302	302
$\bar{B}$	254.95	255.14	255.44	255.48	255.81
P (%)	49.79	49.83	49.89	49.89	49.96
ΔB	11.5	11.64	11.37	11.34	11.36
ΔP (%)	2.24	2.27	2.22	2.21	2.21

Table 10
Statistical results for 1024-bit hash values generated under tests of type A

	N=256	N=512	N=1024	N=2048	N=10,000
$B_{\min}$	474	464	464	456	456
$B_{\max}$	561	562	562	482	483
$\bar{B}$	511.87	511.77	511.72	511.94	511.96
P (%)	49.98	49.97	49.97	49.99	50
ΔB	14.08	15.41	15.7	16.03	16.04
ΔP (%)	1.37	1.5	1.53	1.56	1.57

Table 11
Comparison of statistical results for 128-bit hash values and N=10,000, under tests of type A

	$B_{\min}$	$B_{\max}$	$\bar{B}$	P (%)	ΔB	ΔP (%)
BentSign	44	85	63.99	49.99	5.61	4.38
Ref. [15]	44	84	63.95	49.96	5.62	4.39
Ref. [38]	45	89	64	50.01	5.6	4.37
Ref. [41]	42	83	63.986	49.988	5.616	4.388

In the type B test, the plain data M of size $L = 50n$ bits is generated at random and its corresponding n-bit hash value is computed. Then, a single bit of the plain data is chosen, modified to zero if it is one or to one if it is zero, and the hash value of the modified data is calculated. The two hash values are compared, and the number of changed bits is calculated and recorded as B_i. The same original string is used for all N iterations.

In Fig. 5, the Hamming distance obtained from test type A with $n = 128$ and $N = 10000$ is shown.

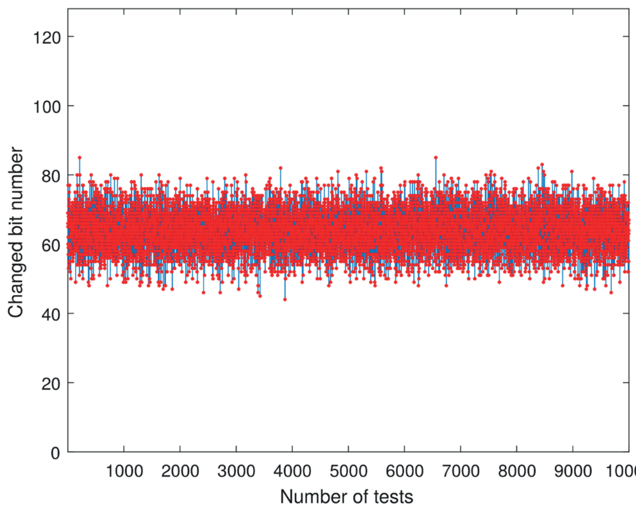

Fig. 5. Distribution of the Bi values obtained in a type A test with $n = 128$ and $N = 10000$

Tables 12–16 list the results obtained in tests of type B for $n = 128, 160, 256, 512, 1024$, and various values of N.

Table 12
Statistical results for 128-bit hash values generated under tests of type B

	N=256	N=512	N=1024	N=2048	N=50×128
$B_{\min}$	49	49	49	44	44
$B_{\max}$	78	80	80	83	83
$\bar{B}$	63.98	64.19	64.67	64.43	64.09
P (%)	49.98	50.15	50.52	50.34	50.07
ΔB	6.24	6.58	6.04	5.99	5.63
ΔP (%)	4.87	5.14	4.72	4.68	4.4

Table 13
Statistical results for 160-bit hash values generated under tests of type B

	N=256	N=512	N=1024	N=2048	N=50×160
$B_{\min}$	66	65	65	63	59
$B_{\max}$	97	100	103	103	103
$\bar{B}$	80.08	80.72	80.46	80.154	80.06
P (%)	50.05	50.45	49.29	50.09	50.04
ΔB	7.17	7.35	6.58	6.38	6.27
ΔP (%)	4.48	4.59	4.11	3.99	3.91

Comparing the results with three chaos based hash algorithms given in Table 17, the BentSign has analogous values.

In Tables 6–16 we can see 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 results illustrate that the BentSign algorithm has strong capacity against confusion and diffusion attacks.

Table 14
Statistical results for 256-bit hash values generated under tests of type B

	N=256	N=512	N=1024	N=2048	N=50×256
$B_{\min}$	109	105	105	105	96
$B_{\max}$	153	153	153	153	155
$\bar{B}$	128.35	129.38	128.58	128.73	128.145
P (%)	50.13	50.14	50.22	50.28	50.05
ΔB	9.34	8.22	7.96	7.81	7.98
ΔP (%)	3.64	3.21	3.04	3.05	3.12

Table 15
Statistical results for 512-bit hash values generated under tests of type B

	N=256	N=512	N=1024	N=2048	N=50×512
$B_{\min}$	227	227	227	221	216
$B_{\max}$	285	291	291	292	294
$\bar{B}$	256.03	256.18	256.82	256.53	255.97
P (%)	50	50.03	50.16	50.1	49.99
ΔB	10.43	10.43	10.43	10.47	11.26
ΔP (%)	2.03	2.03	2.01	2.04	2.19

Table 16
Statistical results for 1024-bit hash values generated under tests of type B

	N=256	N=512	N4=102	N=2048	N=50×1024
$B_{\min}$	470	463	463	448	449
$B_{\max}$	542	553	553	565	565
$\bar{B}$	509.92	507.36	509.37	511.12	510.13
P (%)	49.79	49.54	49.74	49.91	49.92
ΔB	16.19	17.26	17.43	16.83	16.89
ΔP (%)	1.58	1.68	1.7	1.64	1.66

Table 17
Comparison of statistical results for 128-bit hash values and N=2048, under tests of type B

	$B_{\min}$	$B_{\max}$	$\bar{B}$	P (%)	ΔB	ΔP (%)
BentSign	44	83	64.43	50.34	5.99	4.68
Ref. [11]	48	83	64.22	50.17	5.65	4.42
Ref. [12]	47	84	63.94	49.95	5.69	4.44
Ref. [38]	43	43	64.11	50.08	5.59	4.37

4.5. Collision analysis. A general feature of a hash algorithm is to possess collision resistance capability, the following two types of tests are performed, type A and type B [10]. In tests of type A, an original string of size $L = 50n$ is generated and its corresponding n-bit hash code is calculated and stored in ASCII format. Then, a new string is generated by choosing a single bit at random from the input string and modified to zero if it is one or to one if it is zero. The n-bit hash code of the new string is calculated and stored 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. Moreover, the absolute difference D between the two hash codes is calculated by the following formula $D = \sum_{i=1}^{n/8} \left| decimal(e_i) - decimal(e_i') \right|$, where e_i and e_i' be the i-th entry of the input and new hash value, respectively, and function $decimal()$ converts the entries to their equivalent decimal values. The test of type A is repeated $N = 10,000$ times, and experimental minimum, maximum, and mean of D are presented in Table 18 for different hash values of size $n = 128$, 160, 256, and 512.

Table 18
Absolute difference D for hash values generated under tests of type A, where $N = 10,000$

n	Maximum	Minimum	Mean
128	2450	573	1362
160	2744	791	1709
256	4007	1543	2731
512	7275	3712	5459

Table 19 shows the absolute differences of 128-hash codes generated under tests of type A, where $N = 10,000$, and related hash algorithms. The results illustrate that BentSign has comparable values.

Table 19
Comparison of the absolute difference for 128-hash values generated under tests of type A, where $N = 10,000$

n	Maximum	Minimum	Mean
BentSign	2450	573	1362
Ref. [10]	2391	656	1364
Ref. [38]	2386	537	1367
Ref. [41]	2064	655	1367

The count of locations where the ASCII characters are equal, where $N = 10,000$ and the hash codes are calculated under tests of type A, is presented in Table 20 and distributions of the 128-hash and 160-hash values, are illustrated in Figure 6 and Figure 7.

Table 20
Count of positions in Collision test for $N = 10,000$, generated under tests of type A

n	0	1	2	3	4	5
128	9373	607	20	0	0	0
160	9290	688	21	1	0	0
256	8809	1126	60	4	1	0
512	7782	1949	251	17	1	0

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

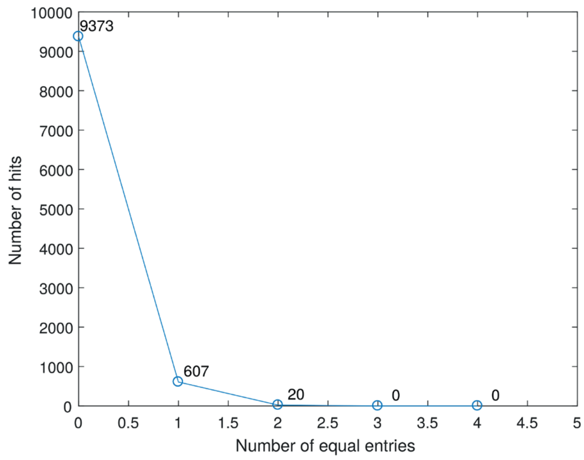

Fig. 6. Distribution of the number of locations where the ASCII symbols are equal in the 128-bit hash values generated under tests of type A, where $N = 10,000$

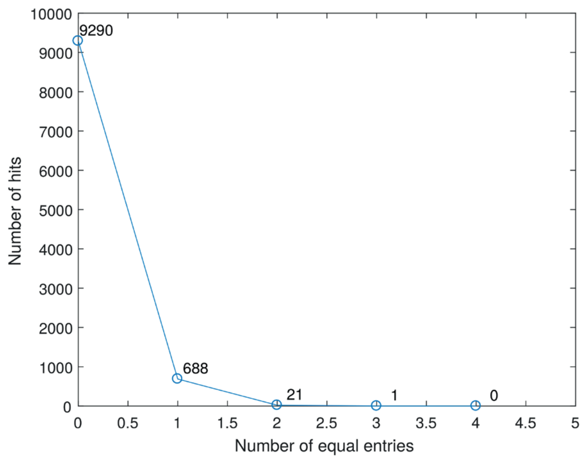

Fig. 7. Distribution of the number of locations where the ASCII symbols are equal in the 160-bit hash values generated under tests of type A, where $N = 10,000$

value is computed. Then, a single bit of the input message is chosen, modified to 0 if it is 1 or to 1 if it is 0, and the hash value of the modified message is calculated. Table 21 presents minimum, maximum, and mean values of D for different hash values of size $n = 128$, 160, 256, and 512 generated under tests of type B. The same input message is used for all N iterations. Comparison with other algorithm is presented in Table 22.

Table 21
Absolute difference D for hash values generated under tests of type B, where $N = 50n$

n	Maximum	Minimum	Mean	N
128	2225	658	1382	6400
160	2533	886	1677	8000
256	4132	1871	2953	12800
512	7224	3687	5345	25600

Table 22
Comparison of absolute difference D for 128-hash values generated under tests of type B, where $N = 6400$

n	Maximum	Minimum	Mean
BentSign	2225	658	1382
Ref. [10]	2421	735	1576
Ref. [11]	2294	661	1360
Ref. [38]	2035	636	1248

In addition to the above experiments, the tests of type B are repeated for very short input data consisting of a single $n-bit$ block, Table 23. The count of positions where the ASCII symbols are equal, generated under tests of type B, is listed in Table 24 and distribution of 160-hash values and $N = 8000$, is illustrated in Figure 8.

Table 23
Absolute difference D for hash values generated under tests of type B, where $N = n$

n	Maximum	Minimum	Mean	N
128	1983	942	1456	128
160	2409	1301	1802	160
256	3821	2327	2977	256
512	6699	4125	5306	512

Table 24
Count of positions in Collision test, generated under tests of type B

n	N	0	1	2	3
128	128	128	0	0	0
128	6400	6021	356	23	0
160	8000	7497	489	14	0
256	12800	11460	1286	54	0

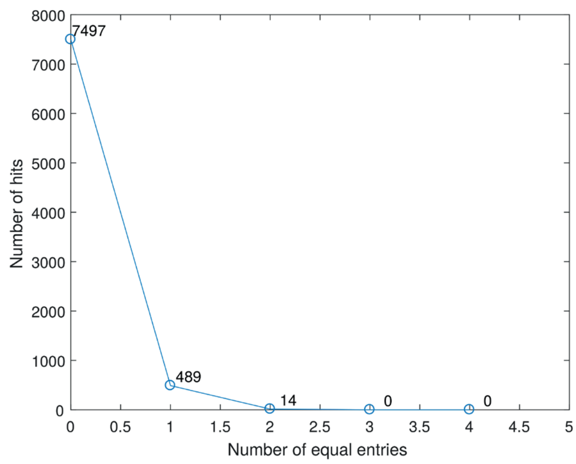

Fig. 8. Distribution of the number of locations where the ASCII symbols are equal in the 160-bit hash values generated under tests of type B, where $N = 8000$

The empirical values confirm that the novel hash algorithm BentSign has an excellent collision resistance.

4.6. Birthday attack analysis. The birthday attack is one of the typical attack on the hash algorithms. The primary target of this attack is to find two input messages with identical hash values [18, 27], and [41]. The time complexity of the attack is $O(2^{n/2})$. Thus for the minimal 128-bit hash value, the attack needs to have 2^{64} attempts, which is sufficient to make the birthday attack unrealistic.

4.7. Meet-in-the-middle attack analysis. The meet-in-the-middle attack attempts to find collisions on intermediate hash values [18]. In the BentSign, the intermediate results are based on the dynamical values of the chaotic attractor. It means that any small differences in the initial values contributes significant changes in the next-state values [24, 27]. As a result, the proposed hash algorithm can prevent meet-in-the-middle attack.

4.8. Second pre-image attack analysis. For any input message M_1, the second pre-image attack tries to find another input message M_2, that has the same hash value [3, 22]. In the proposed hash algorithm, the calculations are based on the mix of the highly nonlinear Boolean function and chaotic values of the Signature attractor. Thus, the BentSign is protected against second pre-image attack.

5. Conclusions

In this paper, we propose a novel keyed hash algorithm based on a Boolean function and a chaotic attractor. The hash algorithm called BentSign is based on classical bent Boolean function and two Signature attractors. The provided theoretical and experimental results verify that the proposed hash algorithm can generate output hash values with a good level of security, strong collision resistance, and protection against most common attacks.

Acknowledgements. This study was funded 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); and the Institute of Telecommunications, Warsaw University of Technology, through the Statutory Grant of the Polish Ministry of Science and Higher Education.

References

[1] A. Akhavan, A. Samsudin, and A. Akhshani, “A novel parallel hash function based on 3D chaotic map”, *EURASIP Journal on Advances in Signal Processing* 2013, 2013:126 (2013).

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

[3] E. Andreeva, C. Bouillaguet, O. Dunkelman, P.A. Fouque, J. Hoch, A. Shamir, J. Kelsey, and S. Zimmer, “New second-preimage attacks on hash functions”, *Journal of Cryptology* 29, 657–696 (2016).

[4] M.S. Baptista, “Cryptography with chaos”, *Physics Letters A* 240 (1-2), 50–54 (1998).

[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] A. Belazi, A. El-Latif, A. Diaconu, R. Rhouma, and S. Belghith, “Chaos-based partial image encryption scheme based on linear fractional and lifting wavelet transforms”, *Optics and Lasers in Engineering* 88, 37–50 (2017).

[7] T.W. Cusick and P. Stănică, *Cryptographic Boolean Functions and Applications*, Academic Press, 2009.

[8] W. Diffie and M.E. Hellman, “New directions in cryptography”, *IEEE Transactions on Information Theory* 22 (6), 644–654 (1976).

[9] K. Ganesan and K. Murali, “Image encryption using eight dimensional chaotic cat map”, *The European Physical Journal Special Topics* 223 (8), 1611–1622 (2014).

[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, H. Yahyaoui, and M. Almulla, “Keyed hash function based on chaotic map”, *Information Sciences* 186, 249–264 (2012).

[13] K. Kordov, “Signature attractor based pseudorandom generation algorithm”, *Advanced Studies in Theoretical Physics* 9 (6), 287–293 (2015).

[14] K. Kordov and L. Bonchev, “Using circle map for audio encryption algorithm”, *Mathematical and Software Engineering* 3 (2), 183–189 (2017).

[15] 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).

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

[17] M. Melosik, P. Sniatala, and W. Marszalek, “Hardware Trojans detection in chaos-based cryptography”, *Bull. Pol. Ac.: Tech.* 65 (5), 725–732 (2017).

[18] A.J. Menezes, P.L. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Boca Raton, 1996.

[19] W. Oniszczuk, “Loss tandem networks with blocking – a semi-Markov approach”, *Bull. Pol. Ac.: Tech.* 58 (4), 673–681 (2010).

[20] M.A. Murillo-Escobar, C. Cruz-Hernández, L. Cardoza-Avendaño, and R. Méndez-Ramírez, “A novel pseudorandom number generator based on pseudorandomly enhanced logistic map”, *Nonlinear Dynamics* 87 (1), 407–425 (2017).

[21] M. Y. Mohamed Parvees, J. Abdul Samath, and B. Parameswaran Bose, ‘ ‘Medical images are safe – an enhanced chaotic scrambling approach”, *Journal of Medical Systems* 41, 167 (2017).

[22] J. Peng, S. Jin, H. Liu, and W. Zhang, “A novel hash function based on hyperchaotic Lorenz system”, *Advances in Intelligent and Soft Computing* 62, 1529–1536 (2009).

[23] K. Pommerening, “Fourier analysis of Boolean maps – a tutorial” 2005, http://www.staff.uni-mainz.de/pommeren/.

[24] 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).

[25] O.S. Rothaus, "On "bent" functions", *Journal of Combinatorial Theory, Series A* 20 (3), 300–305 (1976).

[26] C.H. Skiadas and C. Skiadas, *Chaotic modelling and simulation: analysis of chaotic models, attractors and forms*, CRC Press, 2008.

[27] D.R. Stinson, *Cryptography: theory and practice*, CRC Press, 1995.

[28] B.P. Stoyanov, "Chaotic cryptographic scheme and its randomness evaluation", in *4th AMiTaNS'12*, AIP CP 1487, 397–404 (2012), DOI: 10.1063/1.4758983.

[29] B.P. Stoyanov, "Pseudo-random bit generator based on Chebyshev map", *in 5th AMiTaNS 13*, AIP CP, 1561, 369–372 (2013), DOI: 10.1063/1.4827248.

[30] B.P. Stoyanov, "Pseudo-random bit generation algorithm based on Chebyshev polynomial and Tinkerbell map", *Applied Mathematical Sciences* 8 (125), 6205–6210 (2014), DOI: 10.12988/ams.2014.48676.

[31] B.P. Stoyanov, "Using Circle map in pseudorandom bit generation", *in 6th AMiTaNS'14*, AIP CP 1629, 460–463 (2014), DOI: 10.1063/1.4902309.

[32] B. Stoyanov and K. Kordov, "Cryptanalysis of a modified encryption scheme based on bent Boolean function and feedback with carry shift register", *AIP Conference Proceedings* 1561, 373–377 (2013), DOI: 10.1063/1.4827249.

[33] B. Stoyanov and K. Kordov, "Novel Zaslavsky map based pseudorandom bit generation scheme", *Applied Mathematical Sciences* 8 (178), 8883–8887 (2014).

[34] 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, 1-4 (2014), DOI: 10.1155/2014/986174.

[35] 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), DOI: 10.12988/astp.2015.5342.

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

[37] K. Szczypiorski, "StegHash: new method for information hiding in open social networks", *International Journal of Electronics and Telecommunications* 62 (4), 347–352 (2016).

[38] M. Todorova, B. Stoyanov, K. Szczypiorski, and K. Kordov, "SHAH: hash function based on irregularly decimated chaotic map", *International Journal of Electronics and Telecommunications* 64 (4), 457–465 (2018), DOI: 10.24425/123546.

[39] D. Valli and K. Ganesan, "Chaos based video encryption using maps and Ikeda time delay system", *The European Physical Journal Plus* 132, 542 (2017).

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

[41] 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).

[42] IEEE CS, 754-2008-IEEE Standard for floating-point arithmetic, DOI: 10.1109/IEEESTD.2008.4610935.