Filomat 33:8 (2019), 2257–2268
https://doi.org/10.2298/FIL1908257L

Published by Faculty of Sciences and Mathematics,
University of Niš, Serbia
Available at: http://www.pmf.ni.ac.rs/filomat

New Pseudo-Random Number Generator Based on Improved Discrete-Space Chaotic Map

Dragan Lambić[a], Mladen Nikolić[b]

[a]*Department for Management of Science and Technology Development, Ton Duc Thang University, Ho Chi Minh City, Vietnam*
Faculty of Mathematics & Statistics, Ton Duc Thang University, Ho Chi Minh City, Vietnam
[b]*University of Belgrade, Faculty of Mathematics, Studentski Trg 16, Belgrade, Serbia*

Abstract. In this paper, a new pseudo-random number generator (PRNG) based on improved one-dimensional discrete-space chaotic map is proposed. Like the original, the improved map relies on bijective mapping of permutations and natural numbers. Instead of using standard Lehmer code, we use a mapping computable in linear time, which significantly speeds up the PRNG. Results of NIST 800-22 test suite and TestU01 test suite confirm that the proposed approach can be used for generation of pseudo-random numbers. Due to discrete nature of used chaotic map, the proposed PRNG is not influenced by dynamical degradation and has virtually unlimited key space. Proposed approach has much better ratio between required memory and security level than previous secure one-dimensional discrete-space chaotic PRNGs. Also, proposed PRNG is much faster than other secure PRNGs of the same type. Satisfactory speed and small memory requirements indicate that proposed PRNG has properties desirable for use in devices with limited memory space, such as wireless sensor networks.

1. Introduction

In many fields, such as simulation, numerical analysis, computer programming, decision making, entertainment and cryptography, random input is needed [1]. Generation of random numbers is particularly important in cryptography, because random numbers are used in various phases of cryptographic algorithms for generation of cryptographic keys, initialization vectors for symmetric encryption, passwords, nonce generation etc [2]. Users of cryptographic systems should have access to perfect randomness in order to achieve high level of security, but this is not possible in most realizations of contemporary ciphers. For this reason, great number of cryptographic systems are based on Pseudo-Random Number Generators (PRNGs). Some of the widely used PRNGSs are Mersenne Twister [3], Blum Blum Shub [4] and Linear congruential generator [5]. However, many of the proposed PRNGs can not achieve sufficient randomness, or have other issues, which causes serious security problems of the dependent cryptosystems. For example, output of the Mersenne Twister can be predicted based on observation of 624 iterates [6].

In recent years, there were many examples of using chaos for generation of random numbers. Unfortunately, most of existing chaos-based PRNGs and image encryption algorithms can not achieve sufficient randomness and security due to problems such as inadequate selection of underlying chaotic system [7, 8].

2010 *Mathematics Subject Classification*. 68R01
Keywords. Chaos, Pseudo-random number generator, NIST, Cryptography
Received: 28 July 2018; Revised: 26 January 2019; Accepted: 29 May 2019
Communicated by Marko Petković
Email addresses: dragan.lambic@tdtu.edu.vn (Dragan Lambić), nikolic@matf.bg.ac.rs (Mladen Nikolić)

In many PRNGs secret key is used as the control parameter of the chaotic map. Inappropriate design of its relationship with these parameters might lead to the loss of chaotic properties. There are many examples of cryptographically insecure PRNGs based on chaotic maps.

For example, initial value of chaotic map is used as a secret key in PRNG based on multi-modal maps [9], but unfortunately existence of fixed points in the multi-modal map results in a certain number of weak keys [10]. Inadequate selection of interval for control parameters of quantum chaotic map in PRNG [6] resulted in 99 percent of keys being weak [11]. PRNG based on the Chen chaotic system [12] suffers from dependence among output values of chaotic system which results in a smaller key space [13]. There are some other examples of chaos-based PRNGs in which used chaotic map is not the main cause of low security, but inadequate design of a PRNG. For example, initial condition and control parameter of the PRNG based on pseudorandomly enhanced logistic map [14], are obtained from the secret key. However, great number of different values of secret key produce same initial conditions of chaotic map due to inadequate design of this PRNG. Therefore, there is approximately 2^{50} equivalent secret keys, which makes this PRNG non-resistant to brute-force attack [15].

Low-dimensional chaotic maps have certain advantage over high-dimensional chaotic maps due to their characteristics desirable for cryptography such as simple structure and relatively easy implementation [14]. On the other hand, low-dimensional chaotic maps can be susceptible to some attacks which are not applicable on high-dimensional chaotic maps. For example, simple one-dimensional chaotic maps are vulnerable to the attacks based on nonlinear prediction method [16]. These PRNGs are usually tested for randomness, often with positive outcome. However, this does not necessarily mean that their use in cryptographic applications is safe [8].

For above mentioned reasons, it is desirable to develop new PRNGs based on one-dimensional chaotic maps which have high security level, sufficient for the use in cryptographic systems, which corresponds to the length of the secret key. Recently, PRNG based on one-dimensional discrete-space chaotic map was proposed [17]. This PRNG can achieve very high security level which almost corresponds to the length of the used key, but this security is gained at the expense of speed. In this paper, new PRNG based on improved one-dimensional discrete-space chaotic map [18] is proposed, which has very high security level and is considerably faster than previous PRNG [17].

Main contributions of this paper are:

- Proposed PRNG is based on new chaotic map [18] which is not used before in PRNGs of similar type. Consequently, effect of using this map for generation of pseudo-random numbers are not previously analyzed.
- Speed of chaotic map [18] is significantly improved by using different bijective coding of permutations.
- New fully discrete PRNG is proposed which preserves all advantages of its predecessor [17] but at the same time increases speed of generation without a negative influence on security which was a disadvantage of previous PRNG of this type.

The organization of the paper is the following. In Section 2, an improved composition based discrete chaotic map, is presented. New PRNG is proposed in Section 3. Randomness of the proposed PRNG is analyzed in Section 4. Evaluation of performance and security of the proposed PRNG is conducted in Section 5. Conclusions are drawn in Section 6.

2. Improved composition based discrete chaotic map

Let $P = p_0p_1...p_{n-2}p_{n-1}$ be a permutation of elements $0, 1, ..., n-1$ and $P^r = p_{n-1}p_{n-2}...p_1p_0$ its reverse permutation.

Two permutations f and g of the same elements can be composed, yielding a new permutation $h = f \circ g$ such that for each x it holds $h(x) = f(g(x))$.

We consider permutations of elements $0, \ldots, n-1$. Denote the set of all such permutations by S_n. Permutations can be represented by integers using a bijective mapping $l : S_n \to \{0, 1, 2, ..., n! - 1\}$ called Lehmer code [19], which is defined by:

$$l(P) = \sum_{0 \le i < n} c_i \cdot (n - 1 - i)!$$

where $P \in S_n$ and c_i is the cardinality of $\{j | i > j \wedge p_i < p_j\}$. The computation of Lehmer code and its inverse l^{-1} is described by Algorithms 1 and 2. Division used in 2 is integer division.

```
Input: Permutation $P = p_1 p_2, \ldots, p_{n-1}$
Output: Lehmer code $l$

$l \leftarrow 0$
for $i = 0, \ldots, n-1$ do
    $c_i \leftarrow |\{j | i > j \wedge p_i < p_j\}|$
    $l \leftarrow l + c_i \cdot (n - 1 - i)!$
end
```

Algorithm 1: Algorithm for computation of the Lehmer code

```
Input: Number $x$ from $\{0, \ldots, n! - 1\}$
Output: Permutation $l^{-1}(x) = p_0 p_1 \ldots p_{n-1}$

for $i = 0, \ldots, n-1$ do
    $c_i \leftarrow (x \; mod \; (n - i)!)/(n - 1 - i)!$
    $q_i \leftarrow i$
end
for $i = 0, \ldots, n-1$ do
    $p_i \leftarrow q_{c_i}$
    for $i = c_i, \ldots, n-2$ do
        $q_i \leftarrow q_{i+1}$
    end
end
```

Algorithm 2: Algorithm for computation of the inverse of the Lehmer code

Let $\lfloor y \rfloor$ denote a function which maps a real number y to the largest integer less than or equal to y. Let $\sin(y)$ denote implementation of trigonometric sine function in C++ programming language. Let $\pi = 3.1415926535897932$ denote the constant which represents the value of π rounded to 16 decimal places.

In paper [20] the original version of one-dimensional discrete chaotic map is proposed by

$$X_{i+1} = X_i \circ f(X_i, C) \tag{1}$$

where $X_i, C \in S_n$ and $f : S_n \to S_n$. For $x_i = l(X_i)$ and $c = l(C)$, we obtain:

$$x_{i+1} = l[l^{-1}(x_i) \circ f(l^{-1}(x_i), l^{-1}(c))] \tag{2}$$

where $x_i, c \in \{0, 1, 2, ..., n! - 1\}$ and $f : S_n \to S_n$. In the original version of chaotic map special case $f(X_i, C) = l^{-1}(|l(C \circ X_i) - l((C \circ X_i)^r)|)$ is used [20]. Afterwards, improvement of this chaotic map is proposed in paper [18]. In this paper, we use the special case proposed in paper [18] where

$$f(x_i, c) = l^{-1}(\lfloor \sin(\frac{\pi}{2}(\frac{x_i}{n! - 1} + \frac{c + 1}{n! + 1})) \cdot (n! - 1) \rfloor). \tag{3}$$

Combining (1) and (3) we obtain map $F_n : \{0, 1, 2, ..., n! - 1\} \rightarrow \{0, 1, 2, ..., n! - 1\}$ defined by:

$$F_n(x) = l(l^{-1}(x_i) \circ l^{-1}(\lfloor \sin(\frac{\pi}{2}(\frac{x_i}{n! - 1} + \frac{c + 1}{n! + 1})) \cdot (n! - 1) \rfloor)). \tag{4}$$

In order to further improve this chaotic map from the aspect of speed, in this paper we propose use of different bijective coding instead of Lehmer code. Computation time of Lehmer code according to its definition is quadratic in n. Since our map only requires a bijection between sets S_n and $\{0, 1, \dots, n!-1\}$, which need not be specifically the Lehmer code, we substitute it by a different bijective coding of permutations which works in linear time [34], which will speed up the computation of the map. Coding and decoding of permutation from paper[34] are described by Algorithms 3 and 4.

```
Input: Permutation $P = p_1 p_2, \dots, p_{n-1}$
Output: Permutation code $l$

$l \leftarrow 0$ $m \leftarrow 1$
for $i = 0, \dots, n - 1$ do
    $q_i \leftarrow i$ $r_i \leftarrow i$
end
for $i = 0, \dots, n - 1$ do
    $l \leftarrow m \cdot r_{p_i}$
    $m \leftarrow m \cdot (n - i)$
    $r_{q_{n-1-i}} \leftarrow r_{p_i}$
    $j \leftarrow r_{p_i}$
    $q_j \leftarrow q_{n-1-i}$
end
```

Algorithm 3: Algorithm for computation of the permutation code according to [34]

```
Input: Number $x$ from $\{0, \dots, n! - 1\}$
Output: Permutation $l^{-1}(x) = p_0 p_1 \dots p_{n-1}$

for $i = 0, \dots, n - 1$ do
    $q_i \leftarrow i$
end
for $i = 0, \dots, n - 1$ do
    $j \leftarrow x \ mod \ (n - i)$
    $p_i \leftarrow q_j$
    $q_j \leftarrow q_{n-1-i}$
    $x \leftarrow x/(n - i)$
end
```

Algorithm 4: Algorithm for permutation decoding according to [34]

In order to demonstrate the manner in which this map functions, simple example of one iteration is provided. Let $n = 8$, $c = 1$ and $x_0 = 29$. By inverse bijective coding [34] $x_0 = 29$ is converted to permutation of $n = 8$ elements $X_0 = (5, 3, 0, 7, 4, 6, 2, 1)$. Based on equation 3 we obtain that $f(x_0, c) = l^{-1}(48)$ that is $f(X_0, C) = (0, 6, 7, 5, 4, 3, 2, 1)$. Based on equation 1 we obtain $X_1 = (3, 5, 0, 1, 4, 2, 7, 6)$ which is converted by bijective coding [34] to the value of first iteration $x_1 = 35659$.

Although the chaotic behaviour of the original and improved maps, demonstrated in papers [18, 20], is expected to apply to this version of chaotic map too, further verification is needed in order to demonstrate that change of bijective coding did not deteriorate chaotic behavior. In cobweb plot diagram for

$n = 8, c = 1, x_0 = 29$ presented in Figure 1, qualitative behavior of this version of chaotic map is shown.

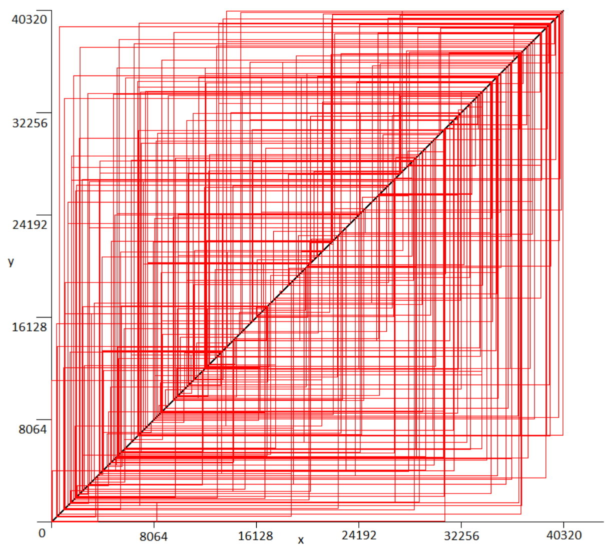

Figure 1: Cobweb plot diagram for $n = 8, c = 1, x_0 = 29$

Values of discrete Lyapunov exponents λ_{F_n} of this version of chaotic map are calculated according to procedure described in [20, 21]. In Table 1, we present minimum, mean, and maximum values of discrete Lyapunov exponents for several values of n. Data from Table 1 show that values of λ_{F_n} are greater than 0 and that they increase when n increases which indicates that this version of the map is also discretely chaotic [21].

Table 1: Discrete Lyapunov exponents

n	min λ_{F_n}	mean λ_{F_n}	max λ_{F_n}
4	1.75	2.01	2.44
5	2.97	3.26	3.48
6	4.58	4.72	4.90
7	6.20	6.32	6.45

The 0-1 test [22] was also conducted. For different values of n, we averaged 1000 values of test statistic K, each computed for a sequence of length 1000 (as recommended by the author of the test implementation). Average values of K statistic for different values of n are presented in Table 2. This test is not completely suitable for smaller values of n because of requirement for 1000 iterations. Space of this discrete-space chaotic map for smaller n is too small (only $n!$ elements) so it is expected that short cycles will occur. This explains why average value of K for $n \leq 7$ is relatively low. For larger n, value of K increases and is very close to ideal value of 1 which indicates that this version of the map is also chaotic [22].

One must note that for values of n greater than 19, standard sine function from C++ programming language should not be used, but a modified version which supports precision corresponding to used integer numbers.

3. The proposed pseudo-random number generator

New pseudo-random number generator is proposed, which relies on composition based discrete chaotic map (eq. 4). It's parameters are the same as for the chaotic map – n, x_0, and c. Values of n lower than 8

Table 2: The 0-1 test results

n	K	n	K
7	0.6385	10	0.9953
8	0.9089	11	0.9977
9	0.9820	12	0.9980

should not be used, since they result in sets S_n of too small cardinality [20]. The proposed pseudo-random number generator is described by the following algorithm:

1. Select the value for n greater than 0. Select the values of x_0 and c from $\{0, 1, 2, ..., n! - 1\}$. Compute $u = \lfloor log_2(n!) \rfloor$ and set i to 0.
2. Assign $v_0 = i$ and $v_{j+1} = v_j/n!$ (integer division is used), for each $0 \leq j \leq J$, where $J = \lfloor log_{n!}(i) \rfloor$.
3. Compute
$$s = \left(x_i + \sum_{j \geq 0}^{J} v_j \right) mod\ n!$$
4. Set $x_{i+1} = F_n(s)$
5. Compute bits $\beta_0\beta_1...\beta_u$ of x_i, bits $\beta'_0\beta'_1...\beta'_u$ of x_{i+1}, and bits $\gamma_0\gamma_1...\gamma_u$ of c.
6. Compute $o_k = \beta_k \oplus \beta'_k \oplus \gamma_k$ for each $0 \leq k \leq u - 4$ (4 most significant bits are discarded).
7. Output sequence o of $u - 3$ bits and increment i by one.
8. If required number of bits is generated, stop. If not, jump to step 2.

4. Randomness analysis

In order to validate whether some PRNG is able to produce pseudo-random sequence, libraries of statistical tests such as NIST and TestU01 test suites are used. Aim of these tests is to establish whether sequences produced by some random number generator suffer from some of the perceived regularities which will distinguish these sequences from ideally random sequences. Therefore, if some PRNG passes these tests successfully, then we can consider that it is resistant to most of attacks based on statistical regularities.

However, such tests have certain limitations and therefore we can not claim that every PRNG which passes these tests is completely secure or that every PRNG which fails these tests is not random. First of all, these tests are based on list of the perceived regularities which is not final, so new tests are progressively developed in order to prevent new attacks. Also, these tests can report false-positives results with much higher probability than expected [23]. Some tests included in NIST test suite always fail when very large number of sequences is tested [23]. TestU01 also suffers from some limitations, most important being that it accepts inputs of only 32 bits. In some cases TestU01 is more sensitive to flaws in the most-significant bits so it is recommended to repeat testing with reversed bits [24].

Also, passing these type of tests does not guarantee security of PRNG from all cryptographical aspects [8]. Further analysis of PRNGs in regards of key space and other aspects of its application in secure cryptographic systems is needed before we can claim that some PRNG is secure. However, statistical testing with test suites such as NIST and TestU01 must be considered as important first step in security analysis of PRNGs.

4.1. NIST 800-22 test suite

A commonly used library of randomness tests for bit sequences is NIST test suite [25]. One way to use NIST library to evaluate randomness of a RNG is to run all statistical tests on several sequences generated by the RNG and to compute the fraction of sequences that pass each test. In order to establish randomness the fraction should be from the interval

$$(1 - \alpha) \pm 3\sqrt{\frac{\alpha}{n}} \tag{5}$$

where $\alpha = 0.01$ and n is the number of sequences. The tests were performed using 1,000 sequences of 1,000,000 bits each. The sequences were generated by the proposed PRNG using parameter value $n = 20$. The results are shown in Table 3.

Table 3: Results of the NIST statistical tests

Test name	Proportion of successful sequences	Result
Approximate entropy	0.989	Success
Frequency within a block	0.992	Success
Cumulative sums	0.989	Success
Discrete Fourier transform	0.990	Success
Frequency	0.990	Success
Longest run in a block	0.991	Success
Non-overlapping template	min 0.981	Success
	max 0.997	Success
Overlapping template	0.987	Success
Random excursions	min 0.982	Success
	max 0.989	Success
Random excursions variant	min 0.986	Success
	max 0.995	Success
Binary matrix rank	0.992	Success
Runs	0.991	Success
Serial	0.993	Success
Linear complexity	0.994	Success
Maurer's universal	0.986	Success

Several tests are actually families of subtests, so in Table 3 the minimum and maximum fractions are given. Those are random excursions (8 tests), random excursions variant (18 tests), and non-overlapping template (148 tests). As the Table 3 shows, the proposed PRNG passed all NIST statistical tests, since the fractions are within the interval given by the equation 5.

4.2. TestU01 test suite

TestU01 is a library of statistical test designed to evaluate randomness of number sequences [28]. Specifically, for binary sequences three batteries of tests are available – Rabbit, Alphabit and BlockAlphabit. In our tests, one sequence of 30,000,000 bits and another one of 1,000,000,000 bits are used. Tests commonly rely on sequence of size of nearly 30,000,000 [29], but we include a longer one for more stringent testing.

In the literature there are various methodologies for interpretation of results of TestU01. In paper [28], in which this test suite was introduced, p-values outside the interval $[10^{-10}, 1-10^{-10}]$ are considered as a complete failures while p-values from intervals $(10^{-10}, 10^{-4}]$ and $[1-10^{-4}, 1-10^{-10})$ are considered as questionable. In paper [30] more stringent bounds for p-values are used. Test with p-value outside the interval $[10^{-9}, 1-10^{-9}]$ is considered as clear failures while p-values from intervals $(10^{-9}, 10^{-3})$ and $(1-10^{-3}, 1-10^{-9})$ required additional testing. Additional testing consists of five retests on different sequences. If p-value from intervals $(10^{-9}, 10^{-3})$ and $(1-10^{-3}, 1-10^{-9})$ is obtained more than once, it is considered as a failure otherwise generator pass test. Values in interval $[10^{-3}, 1-10^{-3}]$ were considered a success.

To generate the sequences, the proposed PRNG was used with parameter values $n = 26, c = 0, x_0 = 0$. The outcomes of the tests are given in Table 4 and they show that the proposed PRNG passes all tests regardless of methodology used to interpret results of testing. However, due to its completely discrete nature and discrete space with variable number of elements which depends on parameter n, for smaller n, space on which chaotic map function is also smaller which can cause occurrence of short orbits. Therefore, there is bigger chance of clear fail when longer sequences are tested for smaller n.

Table 4: Results for the TestU01 statistical tests for bit sequences

Battery	Sequence length	Result
Rabbit	10^9	Success
Rabbit	$3 \cdot 10^7$	Success
Alphabit	10^9	Success
Alphabit	$3 \cdot 10^7$	Success
BlockAlphabit	10^9	Success
BlockAlphabit	$3 \cdot 10^7$	Success

Table 5: Results for the TestU01 statistical tests for floating point number sequences

Battery	Sequence length	Result
SmallCrush	2^{29}	Success
Crush	2^{35}	Success
BigCrush	2^{38}	Success

Although this PRNG is fully discrete and it is not intended for generation of decimal numbers between 0 and 1, but only for generation of bits, we have converted obtained sequence of bits based on parameter values $n = 30, c = 0, x_0 = 0$ into sequence of numbers between 0 and 1 in order to further validate randomness of this generator by testing it with SmallCrush, Crush, and BigCrush batteries. Floating point numbers between 0 and 1 were obtained by generating 32 bits by the generator and dividing the corresponding integer by the range of 32-bit integers. The tests themselves determine the length of the sequences being tested. The outcomes of the tests are given in Table 5 and they show that the proposed PRNG passes all tests.

5. Performance and security analysis

In this section, evaluation of performance and security of the proposed PRNG based on improved one-dimensional chaotic map is presented. Proposed PRNG is compared with existing PRNGs with emphasis on previous approach which uses composition based discrete chaotic map [17]. For comparison with the proposed approach, we choose PRNGs with following properties:

1. they are based on one-dimensional chaotic maps,
2. they are not successfully cryptanalyzed,
3. enough information on their security is available,
4. the key space of at least one of their examples is greater than 2^{128}.

To our knowledge, only chaos based PRNGs [17, 31, 32] meet all the criteria. Unfortunately, the information on some features of some of the PRNGs that we use in comparison is not available. However, for each feature there is at least one generator with available information, which still allows us to evaluate our PRNG against already existing ones.

5.1. Dynamical degradation

Most of the chaotic maps have real domain with infinite precision. However, when these maps are implemented in digital devices with finite precision, problem of dynamical degradation occurs. In order to avoid this problem, some approaches use higher precision for representation of continuous values, but in this way effect of dynamical degradation is only mitigated but not completely solved [17].

PRNG proposed in paper [31] have continuous space domain and is therefore susceptible to dynamical degradation. The proposed PRNG and approaches from papers [17, 32] are all based on discrete-space chaotic maps. For this reason, discretization of continuous values is not used and the process of pseudo-random number generation is not affected by dynamical degradation.

5.2. Key space and security analysis

Secret key of the proposed PRNG consist of initial value x_0 and parameter c. Because x_0 and c are chosen from the set $\{0, 1, 2, ..., n! - 1\}$, the proposed PRNG can generate $n! \cdot n!$ different pseudo-random sequences. Value of n can be any positive integer and therefore key space of the proposed PRNG is virtually unlimited. In PRNGs based on continuous-space chaotic maps, size of key space is based on precision of floating-point numbers which are used as parameters of chaotic maps. PRNGs based on discrete-space chaotic map proposed in [17, 32] have virtually unlimited key space, just like the proposed method. Although PRNGs based on continuous-space chaotic maps usually have limited key space, method presented in [31] also uses permutations of variable size and therefore has virtually unlimited key space.

Although the proposed PRNG and methods from papers [17, 31, 32] all have virtually unlimited key space and consequently virtually unlimited level of security, these methods achieve same key space and security level at different cost of required memory. Example of PRNG presented in paper [32] have 16 rounds of computation and use initial vector (permutation) with $n = 1024$ elements requiring $N = n \cdot log_2(n) = 10240$ bits of memory space in order to achieve key space of 2^{213} pseudo-random sequences. Example of PRNG presented in paper [31] have 18 rounds of computation and use initial vector (permutation) with $n = 2048$ elements requiring $N = 22528$ bits of memory space in order to achieve key space of 2^{400} pseudo-random sequences. We must note that it is assumed that key space of both PRNGs corresponds to their security level.

Security level of PRNG proposed in paper [17] depends on number of output bits b produced in each iteration. When value of b is greater, mentioned PRNG operates faster but with lower security level. When b is smaller, this PRNG has higher level of security but at the expense of speed. If we consider the case when $b = \frac{\lfloor log_2(n!) \rfloor}{2}$ which represents medium level of speed and security, then permutations with $n = 90$ elements and 630 bits of memory space are required in order to achieve security level of approximately 2^{459}. If we consider the case when $b = 1$, then the highest level of security of $2 \cdot n! \cdot (n-1)!$ is achieved, which is nearly the same as key space of $n! \cdot n!$ different pseudo-random sequences. In this case, for PRNG from [17] to achieve key space of roughly 2^{428} distinct bit sequences and, hence, the security level of 2^{428} the permutation of only $n = 50$ elements suffices and such a permutation can be stored within just 300 bits.

Security of the proposed PRNG does not depend on any parameter (except for n) and is equal to its key space. Proposed PRNG also requires only 300 bits for permutation with only $n = 50$ elements in order to achieve key space of approximately 2^{428} which corresponds to its security level.

Based on the above considerations we can conclude that under same memory requirements the proposed method can achieve greater key space and security level than other PRNGs used for comparison. In other words, the proposed approach can achieve same security level with less memory. Compared to most secure version of PRNG [17], proposed approach is slightly better regarding achieved security and required memory, while it is much better compared to medium level secure version of PRNG [17]. In comparison with approaches from papers [31, 32], proposed approach requires significantly less memory in order to achieve similar key space and security level. Low memory requirement is an important advantage of the proposed PRNG in case of devices with small memory. Weak keys do not exist for the proposed PRNG thanks to chaotic map which does not have fixed points [18] and thanks to the mechanism of switching from one orbit to another, present in the design of PRNG, which serves the purpose of avoiding short cycles.

5.3. Key sensitivity analysis

Our assessment of sensitivity with respect to the parameters x_0 and c relies on the Pearson correlation coefficient. The test is performed by generating two pseudo-random sequences $z = [z_0, ..., z_{N-1}]$ and $y = [y_0, ..., y_{N-1}]$ using the same value of one parameter and two neighboring values of the another parameter. The Pearson correlation coefficient is defined by:

$$\rho_{zy} = \frac{\sum_{i=0}^{N-1}(z_i - \overline{z}) \cdot (y_i - \overline{y})}{\sqrt{\sum_{i=0}^{N-1}(z_i - \overline{z})^2} \cdot \sqrt{\sum_{i=0}^{N-1}(y_i - \overline{y})^2}} \qquad (6)$$

where it holds $\overline{z} = \sum_{i=0}^{N-1} \frac{z_i}{N}$ and $\overline{y} = \sum_{i=0}^{N-1} \frac{y_i}{N}$. If the correlation coefficient is 0, meaning no correlation between the sequences, we can conclude that the generator is very sensitive to small changes of parameter values. To test sensitivity of the generator with respect to x_0, we set c to 0, and for x_0, we used values 1000 and 1001 to generate sequences z and y, respectively ($N = 1,000,000$, $n = 20$). Correlation coefficient obtained was $\rho_{zy} = -0.0008$. Therefore, the proposed generator is highly sensible to small changes of x_0. To test sensitivity of the generator with respect to c, we set x_0 to 1000 and for c we used values 0 and 1 are used to generate sequences z and y respectively (($N = 1,000,000$, $n = 20$)). The correlation coefficient obtained was $\rho_{zy} = -0.0009$, demonstrating that the proposed generator is highly sensible to small changes of c, too. References [17, 31, 32] also report high key sensitivity.

5.4. *Cycle length*

PRNGs should have cycles of great length in order to enable smooth functioning in an unspecified period of time which sometimes can be very long. Unfortunately, many of the previous chaos based PRNGs do not have estimated cycle length. Their periodicity was assessed on considerably short bit sequences – of length smaller than 2^{40}. However, cycles should be much longer if PRNG is intended for purposes for which great amount of random data is needed.

Cycle length of the proposed PRNG is approximately $(u-3) \cdot (n!)^J$ where $J = \lfloor log_{n!}(i) \rfloor$. For example, if $n = 128$, $u = 61$ and $J = 28$, the proposed PRNG can produce cycle length of approximately 2^{20059} bits by using only 896 bits of memory space. PRNG [17] can achieve similar cycle length. In order to achieve the cycle length of $2^{19937} - 1$ Mersenne Twister needs 624 words of memory for its seeds [3].

5.5. *Running speed*

In existing literature, different levels of random number generation speed are achieved. For example commercial quantum random number generator ID Quantique QUANTIS can generate 4Mbits/s [33]. Literature on chaos-based PRNGs reports speed from significantly less than 1 Mbit/s up to few thousands of Mbit/s which are usually measured on processors with about 2.0 GHz. Unfortunately, most PRNGs based on one-dimensional chaotic maps (and many others) either do not have a sufficient level of security or there is no sufficient information which could confirm their safety. For PRNGs [31, 32] running speed is not reported. For this reason we have implemented both generators in order to measure their speed. Examples from papers [31, 32] are used but with smaller number of iterations T due to fact that great number of iterations required by these methods significantly reduces their speed, but we took care that choice of T does not result in very low level of security.

All of our experiments were performed on AMD Opteron 6168 1.9GHz processor machine. PRNG [17] has different security levels and speed depending on parameters n and b. Because this PRNG uses quadratic implementation of Lehmer code, speed decreases for larger n. For fairness, smaller value of n will be used for comparison presented in Table 6. Three versions of this PRNG with different security levels will be considered. Medium level of security is achieved for $b = 10$, for which PRNG [17] has speed 3.36 Mbit/s. At lower security level, for $b = 20$, PRNG [17] can generate 6.65 Mbit/s. When security is priority, for $b = 1$, PRNG [17] can generate 0.34 Mbit/s.

The proposed PRNG for $n = 10$ can achieve speed of 24.45 Mbit/s, which is more than 3.6 times faster than least secure (and fastest) version of PRNG [17]. For larger n, proposed PRNG achieves even greater speed. For $n = 15$, proposed PRNG generates 31.27 Mbit/s, while for $n = 20$ it generates 34.18 Mbit/s. The proposed PRNG is 70 times faster than most secure version of PRNG [17], which has similar security level.

The data from Table 6. shows that proposed PRNG is much faster than methods from papers [17, 31, 32] and has satisfactory ratio between speed and security.

6. Conclusion

Simple structure and possibility of relatively easy implementation of low-dimensional chaotic maps are desirable characteristics in cryptography. Unfortunately, great number of chaos-based PRNGs do not have satisfactory level of security in order to be used for cryptographic purposes. Therefore, development of

Table 6: Comparison of reported speeds of chaotic PRNGs

	n	Security level	Speed (Mbit/s)
PRNG in [32] (T=9)	1024	2^{120}	1.99
PRNG in [31] (T=5)	2048	2^{111}	5.36
Least secure PRNG in [17]	10	2^{2}	6.65
Medium secure PRNG in [17]	10	2^{22}	3.36
Most secure PRNG in [17]	10	2^{41}	0.34
The proposed PRNG	10	2^{42}	24.45
The proposed PRNG	15	2^{80}	31.27
The proposed PRNG	20	2^{122}	34.18

PRNG based on low-dimensional chaotic maps with high security level is interesting subject. There are previous examples of secure PRNGs, but high security in these approaches usually comes at the expense of some other important characteristics such as speed or required memory.

In this paper, a new PRNG based on improved one-dimensional discrete-space chaotic map is proposed. Instead of using standard Lehmer code, we use a mapping computable in linear time, which significantly speeds up the PRNG. Results of NIST 800-22 test suite and TestU01 test suite verify randomness of the sequences generated by the proposed PRNG. The proposed PRNG preserves all good characteristics of previous discrete-space approaches such as resistance to dynamical degradation, virtually unlimited key space and great cycle length while improving problematic features of this type of PRNGs.

The proposed PRNG has much better ratio between required memory and security level, compared to previous examples of secure one-dimensional discrete-space chaotic PRNGs. Also, the proposed PRNG achieves much greater speed compared to other PRNGs of this type with same security level. Satisfactory speed and small memory requirements in combination with high security make proposed PRNG usable in wireless sensor networks and other devices with limited memory space.

Acknowledgement

Second author was supported by Serbian Ministry of Science grant 174021.

7. Declarations of interest

Conflict of Interest: The authors declare that they have no conflict of interest.

References

[1] X.Y. Wang, X. Qin, A new pseudo-random number generator based on CML and chaotic iteration, Nonlinear Dynamics, 70 (2012) 1589–1592.
[2] Y. Dodis, D. Pointcheval, S. Ruhault, D. Vergniaud, D. Wichs, Security Analysis of Pseudo-Random Number Generators with Input: /dev/random is not Robust. Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security CCS 13, 647 (2013).
[3] M. Matsumoto, T. Nishimura, Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator, ACM Transactions on Modeling and Computer Simulation, 8(1) (1998) 3–30.
[4] L. Blum, M. Blum, M. Shub, A Simple Unpredictable Pseudo-Random Number Generator, SIAM J Comput, 15(2) (1986) 364–383.
[5] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in Fortran 77: The Art of Scientific Computing, (2nd edition), Press Syndicate of the University of Cambridge, Cambridge 1992.
[6] A. Akhshani, A. Akhavan, A. Mobaraki, S.C. Lim, Z. Hassan, Pseudo random number generator based on quantum chaotic map, Commun Nonlinear Sci Numer Simulat, 19 (2014) 101–111.
[7] G. Alvarez, J.M. Amigo, D. Arroyo, S. Li, Lessons learnt from the cryptanalysis of chaos-based ciphers, in: Kocarev Lj, Lian S. (Eds), Chaos-Based Cryptography: Theory, Algorithms and Applications, Springer-Verlag GmbH, 2011, pp. 257-295.
[8] F. Ozkaynak, Brief review on application of nonlinear dynamics in image encryption, Nonlinear Dynamics, 92(2) (2018) 305–313.

[9] M. Garcia-Martinez, E. Campos-Canton, Pseudo-random bit generator based on multi-modal maps, Nonlinear Dynamics, 82 (2015) 2119–2131.

[10] D. Lambić, Security analysis of the pseudo-random bit generator based on multi-modal maps, Nonlinear Dynamics, 91 (2018) 505–513.

[11] D. Lambić, Security analysis and improvement of the pseudo-random number generator based on quantum chaotic map, Nonlinear Dynamics, 94(2) (2018) 1117–1126.

[12] H. Hu, L. Liu, N. Ding, Pseudorandom sequence generator based on Chen chaotic system, Computer Physics Communications 184(3) (2013) 765-768.

[13] F. Ozkaynak, S. Yavuz, Security problems for a pseudorandom sequence generator based on the Chen chaotic system, Computer Physics Communications 184 (2013) 2178–2181.

[14] M.A. Murillo-Escobar, C. Cruz-Hernandez, L. Cardoza-Avendano, R. Mendez-Ramirez, A novel pseudorandom number generator based on pseudorandomly enhanced logistic map, Nonlinear Dynamics, 87 (2017) 407–425.

[15] D. Lambić, Cryptanalyzing a novel pseudorandom number generator based on pseudorandomly enhanced logistic map, Nonlinear Dynamics, 89 (2017) 2255-2257.

[16] K.M. Short, Steps toward unmasking secure communications, International Journal of Bifurcation and Chaos 4(4) (1994) 959-977.

[17] D. Lambić, M. Nikolić, Pseudo-random number generator based on discrete-space chaotic map, Nonlinear Dynamics, 90 (2018) 223–232.

[18] D. Lambić, S-box design method based on improved one-dimensional discrete chaotic map, Journal of Information and Telecommunication, 2(2) (2018) 181–191.

[19] D.H. Lehmer, Teaching combinatorial tricks to a computer. In: Proc. Sympos. Appl. Math. Combinatorial Analysis, Amer. Math. Soc. 10, 179193 (1960).

[20] D. Lambić, A new discrete chaotic map based on the composition of permutations, Chaos, Solitons & Fractals, 78 (2015) 245–248.

[21] L. Kocarev, J. Szczepanski, J.M. Amigo, I. Tomovski, Discrete Chaos Part I: Theory, IEEE Trans. Circuits and Systems I 53, 1300-1309 (2006).

[22] G.A. Gottwald, I. Melbourne, The 0-1 test for chaos: A review, in Chaos Detection and Predictability, Lecture Notes in Physics Vol. 915, edited by C. Skokos , G. A. Gottwald , and J. Laskar, Springer, 2016.

[23] F. Pareschi, R. Rovatti, G. Setti, On statistical tests for randomness included in the NIST SP800-22 test suite and based on the binomial distribution, IEEE Trans. Inf. Forensics Secur. 7(2) (2012) 491-505.

[24] S. Vigna, An experimental exploration of Marsaglias xorshift generators, scrambled, ACM Transactions on Mathematical Software, 42(4) (2016) 30.

[25] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo, A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST special publication 800-22 (2001)

[26] N.A.N. Abdullah, K. Seman, N.M. Norwawi, Statistical Analysis on LBlock Block Cipher. In: International Conference on Mathematical Sciences and Statistics 2013: Selected Papers, pp. 233 - 245, (2013)

[27] J.S. Teh, A. Samsudin, A Chaos-based Authenticated Cipher with Associated Data, Security and Communication Networks. (2017) 9040518.

[28] P. LEcuyer, R. Simard, TestU01: a C library for empirical testing of random number generators, ACM Trans Math Softw, 33 (2007) article 22.

[29] D.G. Marangon G. Vallone, P. Villoresi, Random bits, true and unbiased, from atmospheric turbulence, Sci. Rep. 4 (2014) 5490.

[30] M.E. ONeill, PCG: a family of simple fast space-efficient statistically good algorithms for random number generation, (HMC-CS-2014-0905).

[31] M. Francois, T. Grosges, D. Barchiesi, R. Erra, Pseudo-random number generator based on mixing of three chaotic maps, Commun Nonlinear Sci Numer Simulat, 19 (2014) 887–895.

[32] M. Francois, T. Grosges, D. Barchiesi, R. Erra, A New Pseudo-Random Number Generator Based on Two Chaotic Maps, Informatica, 24(2) (2013) 181–197.

[33] T. Lunghi, J.B. Brask, C.C.W. Lim, Q. Lavigne, J. Bowles, A. Martin, H. Zbinden, N. Brunner, Self-Testing Quantum Random Number Generator, Phys Rev Lett, 114 (2015) 150501.

[34] A. Comeau, Mapping between permutations and natural numbers. Unpublished: http://antoinecomeau.blogspot.rs/2014/07/mapping-between-permutations-and.html, (2014)