Acta Informatica (2023) 60:317–329
https://doi.org/10.1007/s00236-023-00440-w

ORIGINAL ARTICLE

Simple chain automaton random number generator for IoT devices

Pál Dömösi[1,2] · Géza Horváth[1] · Norbert Tihanyi[3,4]

Received: 24 March 2022 / Accepted: 15 May 2023 / Published online: 8 June 2023

Abstract
Random numbers are very important in many fields of computer science. Generating high-quality random numbers using only basic arithmetic operations is challenging, especially for devices with limited hardware capabilities, such as Internet of Things (IoT) devices. In this paper, we present a novel pseudorandom number generator, the simple chain automaton random number generator (SCARNG), based on compositions of abstract automata. The main advantage of the presented algorithm is its simple structure that can be implemented easily for very low computing capacity IoT systems, FPGAs or GPU hardware. The generated random numbers demonstrate promising statistical behavior and satisfy the NIST statistical suite requirements, highlighting the potential of the SCARNG for practical applications.

1 Introduction

Random number generation is very important in many field of computer science, including Monte Carlo simulations [1], cryptography, and other computer applications. The study of pseudorandom number generators (PRNG) has received significant attention from researchers, and various fields of mathematics have been utilized to create random numbers such as number theory [2] or chaos theory [3]. Generating random numbers using automata theory is not new in the literature. Already in 1986 S. Wolfram proposed a method to generate random numbers using cellular automaton [4]. In 1989, Hortensius et al. proposed

✉ Pál Dömösi
domosi@unideb.hu; domosi.pal@nye.hu

Géza Horváth
horvath.geza@inf.unideb.hu

Norbert Tihanyi
ntihanyi@inf.elte.hu; norbert.tihanyi@tii.ae

1 Faculty of Informatics, University of Debrecen, Kassai út 26, Debrecen 4028, Hungary

2 Institute of Mathematics and Informatics, University of Nyíregyháza, Sóstói út 31/B, Hungary, Nyíregyháza 4400, Hungary

3 Department of Computer Algebra, Eötvös Loránd University, Pázmány Péter stny. 1/C, Budapest 1117, Hungary

4 Technology Innovation Institute, AIDRC, Masdar City, Abu Dhabi, United Arab Emirates

random number generation for VLSI systems using 1-dimensional cellular automata [5]. Since then, numerous articles have been published in the literature that build upon various types of cellular automata (see, e.g., [6–10]). The simple construction of cellular automata makes them an attractive option for generating random numbers or developing novel cryptosystems; however, this simplicity also makes them vulnerable to specific types of attacks [11, 12]. Therefore, it is reasonable to explore the creation of novel pseudorandom number generators and encryption schemes rely on automata theory principles different from cellular automata.

The aim of this paper is twofold: to develop a very simple automaton that successfully passes known statistical tests and to create a PRNG with very simple structure that is suitable for use in embedded devices and Internet of Things (IoT) applications. IoT is a ubiquitous and pervasive network of physical objects that are embedded with sensors, software, and other technologies, enabling them to connect and exchange data with other devices and systems over the Internet or other communication networks [13]. These interconnected devices can communicate with each other, collect data, and perform various tasks autonomously; therefore, ensuring the protection of each device is crucial in a complex IoT environment. Compared to classical computer systems, IoT devices used in these environments are often low-cost RFID tags, sensors, contactless smart cards that lack sufficient processing power and memory capabilities. As a result, standard cryptographic primitives may not be suitable for IoT endpoints and performing classical cyber attacks is often easier than on typical computer systems, making them a potential target for malicious users [14]. Lightweight cryptography is often employed to address the security concerns of constrained environments; however, the diverse nature of IoT devices presents a significant challenge in the design of cryptographic primitives for IoT systems.

In this paper, the authors continue their joint research of cryptographic tools based on compositions of abstract finite automata [15, 16]. In 2021, the authors presented a full cycle length counter-based PRNG based on automata compositions [17]. The promising statistical behavior of the proposed construction inspired us to explore the potential for developing even simpler generators based on automata compositions. The primary idea behind that PRNG was that random numbers are generated as state transitions of a given Gluskov product of automata, where the states of this Gluskov product are considered as binary strings of a given length m. To be more precise, for an appropriate fixed state a_0 and fixed input letter x of the considered Gluskov product, the generated pseudorandom numbers are $\delta(a_1, x), \delta(a_2, x), \delta(a_3, x), \dots,$ $\delta(a_n, x)$ where δ denotes the transition function of the composite automaton and $a_{i+1} = a_i + 1$ mod 2^m for every $i = 1, 2, \dots n$. It was shown that the generator produces good-quality numbers and passes well known statistical tests, but a significant drawback of the approach is the complexity of the automata composition being considered. In this paper, we introduce a simpler and more compact PRNG, which we refer to as the *simple chain automaton random number generator (SCARNG)*. The main benefit of the algorithm proposed in this paper is its simple structure which allows easy implementation on FPGAs, GPUs, and IoT devices with minimal computing capacity. The algorithm proposed in this paper has successfully passed the National Institute of Standards and Technology (NIST) statistical test [18]. Furthermore, the unique properties of the introduced chain automaton can open new research directions.

1.1 Definitions and notations

All concepts needed for seamless understanding are presented in this paper. However, this paper does not cover the fundamentals of automata theory and their introductions. For basic notations and further insight into the subject, we recommend the reader to see the SIAM monograph on Algebraic Theory of Automata Networks written by Dömösi and Nehaniv [19].

As usual let us denote by Σ a finite, nonempty set of symbols, which we call alphabet. For any set Σ, let Σ^* represent the free monoid over Σ, which means the set of all strings over an alphabet Σ. Let λ be the unit element of Σ^* called *empty word*. Then, for every $p \in \Sigma^*$, $p\lambda = \lambda p = p$; moreover, for every pair $p, q \in \Sigma^* \backslash \{\lambda\}$, $pq = x_1 \cdots x_n x_{n+1} \cdots x_{n+k}$ whenever $p = x_1 \cdots x_n, q = x_{n+1} \cdots x_{n+k}$ for some $x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+k} \in \Sigma$, and $k, n > 0$. In addition, for every $p \in \Sigma^*$, let $p^0 = \lambda$, and for every positive integer $n \geq 1$, let $p^n = p^{n-1}p$.

By an *automaton*, we mean a deterministic finite automaton without outputs. To be more precise, an automaton is an algebraic structure $\mathcal{A} = (A, \Sigma, \delta)$ consisting of the nonempty and finite *state set* A, the nonempty and finite *input set* Σ, and a *transition function* $\delta : A \times \Sigma \rightarrow A$. The transition matrix of an automaton is represented by a matrix where rows are associated with each input and columns with each state. For any row denoted by input $x \in \Sigma$ and any column denoted by state $a \in A$, the matrix entry contains the state $\delta(a, x)$. If all rows of the transition matrix are permutations of the state set, then we speak about *permutation automaton*.

We shall use the concept of transition function of automata in the usual extended form. Given an automaton $\mathcal{A} = (A, \Sigma, \delta)$, its extended transition function is the mapping $\delta : A \times \Sigma^* \rightarrow A$, where Σ^* denotes the free monoid over Σ, and $\delta : A \times \Sigma \rightarrow A$ is extended to $\delta : A \times \Sigma^* \rightarrow A$ by the following recursive definition: for every state $a \in A$, $\delta(a, \lambda) = a$, and for every triplet $a \in A$, $p \in \Sigma^*$, $x \in \Sigma$, $\delta(a, px) = \delta(\delta(a, p), x)$. A Latin square of order n is an $n \times n$ matrix (with n rows and n columns), where each cell is filled with an element from a set of n elements $\{a_0, a_1, \ldots, a_{n-1}\}$. Each element occurs exactly once in each row and each column of the matrix. In this paper, we will consider a special composition of automata, called key automaton, whose state transition table forms a Latin square. By these properties, the considered pseudorandom number generator has a strong immunity against statistical attacks. Prior to presenting the precise design of our generator, we will first introduce several essential concepts needed to understand the SCARNG construction.

2 Key automaton

Let $\mathcal{A}_i = (A_i, \Sigma_i, \delta_i)$ be automata where $i \in \{1, \ldots, n\}$, $n \geq 1$. Take a finite nonvoid set Σ and a *feedback function* $\varphi_i : A_1 \times \cdots \times A_n \times \Sigma \rightarrow \Sigma_i$ for every $i \in \{1, \ldots, n\}$. A *Gluškov-type product* of the automata $\mathcal{A}_i$ with respect to the feedback functions φ_i ($i \in \{1, \ldots, n\}$) is defined to be the automaton $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n(\Sigma, (\varphi_1, \ldots, \varphi_n))$ with state set $A = A_1 \times \cdots \times A_n$, input set Σ, transition function δ given by $\delta((a_1, \ldots, a_n), x) = (\delta_1(a_1, \varphi_1(a_1, \ldots, a_n, x)), \ldots, \delta_n(a_n, \varphi_n(a_1, \ldots, a_n, x)))$ for all $(a_1, \ldots, a_n) \in A$ and $x \in \Sigma$. In particular, if $\mathcal{A}_1 = \ldots = \mathcal{A}_n$, then we say that $\mathcal{A}$ is a *Gluškov-type power.*

One of the most simple Gluškov-type powers is the following construction. Let $\mathcal{A} = (\Sigma, \Sigma, \delta)$ be an automaton whose set of states coincides with the set of input signals. Given a positive integer n, define the automaton $\mathcal{B} = (\Sigma^n, \Sigma^n, \delta_{\mathcal{B}})$ such that for every pair $(a_1, \ldots, a_n), (x_1, \ldots, x_n) \in \Sigma^n$, $\delta_{\mathcal{B}}((a_1, \ldots, a_n), (x_1, \ldots, x_n)) = (a'_1, \ldots, a'_n)$, where the first component a'_1 of the next-state vector $(a'_1, \ldots, a'_n)$ is a result of a state transition chain of automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ with $\mathcal{A}_1 = \ldots = \mathcal{A}_n = \mathcal{A}$ such that the last member

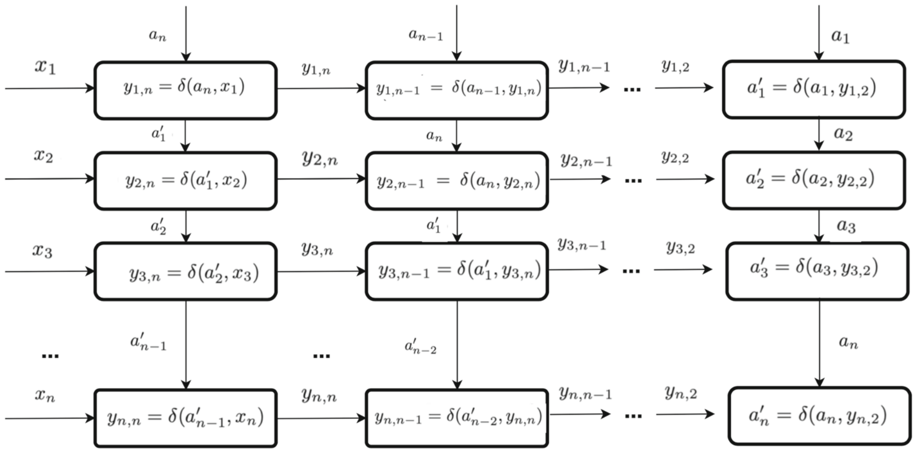
x_1
a_n
$y_{1,n} = \delta(a_n, x_1)$
$y_{1,n}$
a_{n-1}
$y_{1,n-1} = \delta(a_{n-1}, y_{1,n})$
$y_{1,n-1}$
$y_{1,2}$
a_1
$a'_1 = \delta(a_1, y_{1,2})$
a'_1
a_n
a_2
x_2
$y_{2,n} = \delta(a'_1, x_2)$
$y_{2,n}$
$y_{2,n-1} = \delta(a_n, y_{2,n})$
$y_{2,n-1}$
$y_{2,2}$
$a'_2 = \delta(a_2, y_{2,2})$
a'_2
a'_1
a_3
x_3
$y_{3,n} = \delta(a'_2, x_3)$
$y_{3,n}$
$y_{3,n-1} = \delta(a'_1, y_{3,n})$
$y_{3,n-1}$
$y_{3,2}$
$a'_3 = \delta(a_3, y_{3,2})$
a'_{n-1}
a'_{n-2}
a_n
x_n
$y_{n,n} = \delta(a'_{n-1}, x_n)$
$y_{n,n}$
$y_{n,n-1} = \delta(a'_{n-2}, y_{n,n})$
$y_{n,n-1}$
$y_{n,2}$
$a'_n = \delta(a_n, y_{n,2})$

Fig. 1 Chain power state transition

$\mathcal{A}_n$ of the chain goes from the state a_n into its next state under the effect of the input signal component x_1, and each of the previous members of the chain goes from its state $a_i, i = 1, \ldots, n-1$ into its next state under the effect of its input signal which coincides with the result of the state transition of the $(i+1)^{th}$ member of the considered chain of automata. For every further $j = 1, \ldots, n$ we repeat this procedure such that the initial state $(a_1, \ldots, a_n)$ is changed with $(a_j, a_{j+1}, \ldots, a_n, a'_1, \ldots, a'_{j-1})$. In formulas, let $a'_1 = \delta(a_1, \delta(a_2, \ldots, \delta(a_n, x_1) \ldots)), a'_2 = \delta(a_2, \delta(a_3, \ldots, \delta(a_n, \delta(a'_1, x_2)) \ldots))$, $a'_3 = \delta(a_3, \delta(a_4, \ldots, \delta(a_n, \delta(a'_1, \delta(a'_2, x_3))) \ldots)), \ldots, \quad a'_n = \delta(a_n, \delta(a'_1, \ldots,$ $\delta(a'_{n-1}, x_n) \ldots))$. From now on, the above constructed automaton $\mathcal{B}$ is referred to as a *chain power* of $\mathcal{A}$. The diagram illustrating the state transition of the chain power is viewed in Fig. 1.

Proposition 1 *Suppose that $\mathcal{A} = (\Sigma, \Sigma, \delta)$ is a permutation automaton. Then, every chain power of $\mathcal{A}$ also is a permutation automaton.*

Proof Assume that $\mathcal{A} = (\Sigma, \Sigma, \delta)$ is a permutation automaton. Then, by definition, all rows of its transition matrix are permutations of the state set. Therefore, none of these rows contain repetition. Consequently, for any states $a, b \in \Sigma$ and input $x \in \Sigma$, $\delta(a, x) = \delta(b, x)$ implies $a = b$. Given a positive integer n, let $\mathcal{B} = (\Sigma^n, \Sigma^n, \delta_{\mathcal{B}})$ be a chain power of $\mathcal{A}$ and suppose that $\mathcal{B}$ is not a permutation automaton. Then, it has distinct states $(a_1, \ldots, a_n), (b_1, \ldots, b_n) \in \Sigma^n$ and input sign $(x_1, \ldots, x_n) \in \Sigma$ with $\delta_{\mathcal{B}}((a_1, \ldots, a_n), (x_1, \ldots, x_n)) = \delta_{\mathcal{B}}((b_1, \ldots, b_n), (x_1, \ldots, x_n)) = (c_1, \ldots, c_n)$ for some $(c_1, \ldots, c_n) \in \Sigma^n$. Suppose that $i \in \{1, \ldots, n\}$ is the maximal index for which $a_i \neq b_i$. Then, by definition, $c_i = \delta(a_i, \delta(a_{i+1}, \ldots, \delta(a_n, \delta(c_1, \ldots, \delta(c_{i-1}, x_i)))))$ and also $c_i = \delta(b_i, \delta(a_{i+1}, \ldots, \delta(a_n, \delta(c_1, \ldots, \delta(c_{i-1}, x_i)))))$. Put $x = \delta(a_{i+1}, \ldots, \delta$ $(a_n, \delta(c_1, \ldots, \delta(c_{i-1}, x_i))))$ and recall that $\mathcal{A}$ is a permutation automaton. Obviously, then $\delta(a_i, x) \neq \delta(b_i, x)$ contradicting the assumption $c_i = \delta(a_i, x) = \delta(b_i, x)$. Therefore, $\mathcal{B}$ should be a permutation automaton. This completes the proof.

Theorem 2 *Suppose that the transition matrix of an automaton $\mathcal{A} = (\Sigma, \Sigma, \delta)$ forms a Latin square. Then every chain power of $\mathcal{A}$ also has this property.*

Proof Consider an automaton $\mathcal{A} = (\Sigma, \Sigma, \delta)$ and assume that its transition matrix forms a Latin square. Let $\mathcal{B} = (\Sigma^n, \Sigma^n, \delta_\mathcal{B})$ be its chain power for some positive integer n. By Proposition 1, $\mathcal{B}$ is a permutation automaton. Therefore, it is enough to show that for every state $(a_1, \dots, a_n) \in \Sigma^n$ and distinct pair $(x_1, \dots, x_n), (y_1, \dots, y_n)$ of input signals, $\delta_\mathcal{B}((a_1, \dots, a_n), (x_1, \dots, x_n)) \neq \delta_\mathcal{B}((a_1, \dots, a_n), (y_1, \dots, y_n))$. Put $(b_1, \dots, b_n) = \delta_\mathcal{B}((a_1, \dots, a_n), (x_1, \dots, x_n))$ and $(c_1, \dots, c_n) = \delta_\mathcal{B}((a_1, \dots, a_n), (y_1, \dots, y_n))$. Let $i \in \{1, \dots, n\}$ be the minimal index with $x_i \neq y_i$. Then, by definition, either $i = 1$, or $b_1 = c_1, \dots, b_{i-1} = c_{i-1}$. If $i = 1$ then $b_1 = \delta(a_1, \delta(a_2, \dots, \delta(a_n, x_1) \dots))$ and $c_1 = \delta(a_1, \delta(a_2, \dots, \delta(a_n, y_1) \dots))$ with $x_1 \neq y_1$. Recall that the transition matrix of $\mathcal{A}$ forms a Latin square. Obviously, then using that the columns of the transition matrix are also permutations of Σ, we have $\delta(a_n, x_1) \neq \delta(a_n, y_1)$. Inductively, $\delta(a_{n-1}, \delta(a_n, x_1)) \neq \delta(a_{n-1}(a_n, y_1)), \dots, \delta(a_1, \delta(a_2, \dots, \delta(a_n, x_1) \dots)) \neq \delta(a_1, \delta(a_2, \dots, \delta(a_n, y_1) \dots))$. Thus, we are ready if $i = 1$. Otherwise, using $b_1 = c_1, \dots, b_{i-1} = c_{i-1}$, $b_i = \delta(a_i, \delta(a_{i+1}, \dots, \delta(a_n, \delta(b_1, \dots, \delta(b_{i-1}, x_i)))))$ $c_i = \delta(a_i, \delta(a_{i+1}, \dots, \delta(a_n, \delta(b_1, \dots, \delta(b_{i-1}, y_i)))))$. Because the columns of the transition matrix of $\mathcal{A}$ are permutations of Σ, we have $\delta(b_{i-1}, x_i) \neq \delta(b_{i-1}, y_i)$, and inductively, $\delta(b_{i-2}, \delta(b_{i-1}, x_i)) \neq \delta(b_{i-2}, \delta(b_{i-1}, x_i)), \dots$, $\delta(a_{i+1}, \dots, \delta(a_n, \delta(b_1, \dots, \delta(b_{i-1}, x_i)))) \neq \delta(a_{i+1}, \dots, \delta(a_n, \delta(b_1, \dots, \delta(b_{i-1}, y_i))))$. Put $x = \delta(a_{i+1}, \dots, \delta(a_n, \delta(b_1, \dots, \delta(b_{i-1}, x_i))))$, and $y = \delta(a_{i+1}, \dots, \delta(a_n, \delta(b_1, \dots, \delta(b_{i-1}, y_i))))$ and recall again that the columns of the transition matrix of $\mathcal{A}$ are permutations of Σ. Then, we receive $b_i = \delta(a_i, x) \neq \delta(a_i, y) = c_i$. Therefore, $(b_1, \dots, b_n) \neq (c_1, \dots, c_n)$ as we stated. The proof is complete.

Let $\mathcal{A} = (\Sigma, \Sigma, \delta)$ be an automaton whose transition matrix forms a Latin square. Given a positive integer n, consider a chain power $\mathcal{B} = (\Sigma^n, \Sigma^n, \delta_\mathcal{B})$ of $\mathcal{A}$. From now on, we say that $\mathcal{B}$ is a *key automaton*.

3 Counter-based pseudorandom number generator

The use of an integer counter as the sole internal state of a generator is a well-known approach for developing a pseudo-random number generator, known as a counter-based PRNG (see e.g. [20]).

The state transition function is an increment by one modulo the size n of the finite state set $S = \{0, \dots, n-1\}$ and the complexity comes in the map from the state to the random sample. Formally, a counter-based pseudorandom number generator (CBPRNG) is a structure $\mathcal{CBPRNG} = (K, Z_J, S, f, U, g)$, where K is the key space; $Z_J = \{0, 1, ..., J-1\}$, where J is a positive integer called output multiplicity; S is the state space; U is the output space; $f : S \to S$ is the state transition function, $s_i = f(s_{i-1})$; $g : K \times Z_J \times S \to U$ is the output function.

The $\mathcal{CBPRNG} = (K, Z_J, S, f, U, g)$ works in discrete time scale. It starts from a fixed state $s \in S$, called *initial state* and a fixed key $k \in K$. Then, the generated random number sequence is $g(k, 0, f^1(s)), \dots, g(k, J-1, f^1(s)), g(k, 0, f^2(s)), \dots, g(k, J-1, f^2(s)), g(k, 0, f^n(s)), \dots, g(k, J-1, f^n(s))$, where $f^1(s) = f(s)$, $f^2(s) = f(f(s))$ and $f^n(s) = f(f^{n-1}(s))$ for every further $n > 2$. In this case, the vector $(g(k, 0, f(s)), \dots, g(k, J-1, f(s))$ is called the output vector of initial state.

Given a $\mathcal{CBPRNG} = (K, Z_J, S, f, U, g)$, we say that its state transition function $f : S \to S$ *has a full cycle* if for every $s \in S$, $S = \{f^n(s) \mid n = 1, \dots, |S|\}$, where, by definition, $|S|$ denotes the cardinality of S. Moreover, a $\mathcal{CBPRNG}$ is said to have a *full cycle* or *full* *period* if for any key and initial state $s \in S$, the $\mathcal{CBPRNG}$ traverses every output vector $(u_0, \ldots, u_{J-1}) \in U^J$ before returning to the output vector of the initial state.

Suppose that, for the simplicity, S is a set of binary strings of the same fixed length. We say that $\mathcal{CBPRNG}$ is of a *trivial period* if $|S|$ is the minimal positive integer n for which $g(k, i, f^n(s + n + m \mod |S|)) = g(k, i, f^n(s + m \mod |S|)), k \in K, i \in Z_J, m \geq 1$.[1] The following statement is clear.

Proposition 3 *A $\mathcal{CBPRNG} = (K, Z_J, S, f, U, g)$ has a full cycle if and only if its state transition function $f : S \to S$ has a full cycle, and for every key $k \in K$, the function $g_k : S \to U^J$ with $g_k(s) = (g(k, 0, s), \ldots, g(k, J - 1, s)), s \in S$ is bijective.*

If the set Z_J of $\mathcal{CBPRNG} = (K, Z_J, S, f, U, g)$ is a singleton (i.e., $Z_J = \{0\}$), then we will write g in the form $g : K \times S \to U$, and then, we say that $\mathcal{CBPRNG}$ has a *simple output multiplicity*. In this case, we will write also $\mathcal{CBPRNG}$ in the form $\mathcal{CBPRNG} = (K, S, f, U, g)$. For the sake of simplicity, in this paper we consider CBPRNGs having a simple output multiplicity.[2]

Given an output function $g : K \times S \to U$ having a simple output multiplicity and assume that $U \subseteq S$. We say that the output function $g' : K \times S \to U$ is a double round of the output function $g : K \times S \to U$ if for every $k \in K, s \in S$, $g'(k, s) = g(k, g(k, s))$. In general, we say that $g' : K \times S \to U$ is a $t(> 1)$-times round of $g : K \times S \to U$ if for every $k \in K, s \in S$, $g'(k, s) = g(k, h(k, s))$ such that $h : K \times S$ is a $(t - 1)$-times round of $g : K \times S$. Finally, the single round of $g : K \times S \to U$ is the function $g : K \times S \to U$ itself.

For CBPRNGs, we should have that g is complex and f is a simple counter with $f(s) = (s + 1) \mod 2^p$, where p is the state size in bits and $S = \{0, \ldots, 2^{p-1}\}$[20]. Applying the ideas of this construction, in this paper we consider CBPRNGs, where f is a counter, and g is defined by composition of abstract finite automata.

Given a nonempty set Σ, put $\overline{w} = (a_1, \ldots, a_m)$ for every $m > 1$ and $w = a_1 \cdots a_m$ with $(a_1, \ldots, a_m) \in \Sigma^m$. Consider a pair of nonempty sets Δ, Σ, a positive integer $n > 0$ and functions $f : \Delta^n \times \Sigma^n \to \Sigma^n$, $g : \{w \mid \overline{w} \in \Delta^n\} \times \{z \mid \overline{z} \in \Sigma^n\} \to \{w \mid \overline{w} \in \Delta^n\}$. Suppose that for every pair $\overline{w} \in \Delta^n, \overline{z} \in \Sigma^n$, it holds that $f(\overline{w}, \overline{z}) = \overline{g(w, z)}$. Then, we say that g *represents* f in one round. Moreover, if there exists a $t \geq 1$ such that for every pair $\overline{w} \in \Delta^n, \overline{z} \in \Sigma^n$, it holds that $f(\overline{w}, \overline{z^t}) = \overline{g(w, z)}$, then we say that f *represents* g *in* t *rounds*.

Theorem 4 *Given a positive integer $t \geq 1$, every key automaton transition function represents in $t \geq 1$ rounds an output function of a counter-based pseudorandom number generator (having a simple output multiplicity).*

Proof As the proof of our statement, we give a construction of an appropriate counter based PRNG (CBPRNG) $\mathcal{CBPRNG} = (K, S, f, U, g)$ having this property. First of all, consider a counter which realizes the state function as $f(n) = n + 1 \mod m$, where m is a sufficiently large positive integer (preferably $m = 2^{128}$), and n is given as a fixed-length binary number (preferably with 128-bit length).

Thus, the state space is $S = \{0, \ldots, m - 1\}$.

The elements of the state set S of $\mathcal{CBPRNG}$ may be considered binary strings of fixed length. We assume that the state space S of $\mathcal{CBPRNG}$ coincide with $\{\overline{w} \mid w \in \Sigma^n\}$, where

[1] Of course, for every pair $s_1, s_2, s_1 + s_2 \mod |S|$ denotes the element of S having the numeric value of the sum $n_1 + n_2$, where n_1 is the numeric value of s_1 and n_2 is the numeric value of s_2.

[2] By an easy extension, our construction can be extended for the non-simple case.

Σ^n is the input set of $\mathcal{K}$ for an appropriate positive integer $n \geq 1$. Moreover, we also assume that the output set U and also the key space of $\mathcal{CBPRNG}$ coincides with K.

In addition, assume that the output function $g : K \times S \to U$ of $\mathcal{CBPRNG}$ is given as $\overline{g(k,s)} = \delta_{\mathcal{K}}(\overline{k}, \overline{s^t}))$, $k \in K, s \in S$ for every $k \in K$, $s \in S$ and fixed $t \geq 1$. Obviously, then the transition function $\delta_{\mathcal{K}}$ represents the output function g. This completes the proof.

By Proposition 3 and Theorem 4, we can derive the following.

Proposition 5 *Let $\mathcal{CBPRNG} = (K, S, f, U, g)$ be a counter based pseudorandom number generator with simple output multiplicity (i.e., $Z_J = \{0\}$) and assume that the transition function of a given key automaton represents the output function of $\mathcal{CBPRNG}$. If $\mathcal{CBPRNG}$ is of a single round, then it has a full cycle.*

Proof Recall that, by definition, for every key automaton, the transition matrix of its basic automaton forms a Latin square. Therefore, by Theorem 2, the transition matrix of all key automata has this property. Thus, all rows and all columns of their transition matrix of $\mathcal{K}$ are a permutation of its state set.

Of course, because the state transition f of $\mathcal{CBPRNG}$ (having a simple output multiplicity) is a simple counter with $f(s) = (s+1) \mod 2^p$, where p is the state size in bits and $S = \{0, \ldots, 2^{p-1}\}$, f has a full cycle. Moreover, by Theorem 2 the key automaton $\mathcal{K} = (\Sigma^n, \Sigma^n, \delta_{\mathcal{K}}), n > 1$ is a permutation automaton; therefore, for every input sign $x \in \Sigma^n$, $h_x : \Sigma^n \to \Sigma^n$ with $h_x(y) = \delta_{\mathcal{K}}(x, y)$ is a bijective mapping of Σ^n onto itself. Because $t = 1$, $\overline{\delta_{\mathcal{K}}(x, y)} = g(\overline{x}, \overline{y})$. Obviously, then the function $g_{\overline{x}} = g(\overline{x}, \overline{y}), \overline{y} \in \Sigma^n$, where g denotes the output function of $\mathcal{CBPRNG}$, is also bijective. By Proposition 3, that means that $\mathcal{CBPRNG}$ (having a simple output multiplicity) has a full cycle.

Remark 1 Consider a key automaton $\mathcal{K} = (\Sigma^n, \Sigma^n, \delta_{\mathcal{K}})$ such that its transition function represents the output function g of a given $\mathcal{CBPRNG}$ in t rounds. Then, g is of the form $g : \{w \mid \overline{w} \in \Sigma^n\} \times \{z \mid \overline{z} \in \Sigma^n\} \to \{w \mid \overline{w} \in \Sigma^n\}$. Then, the function f_w with $\overline{f_w(z)} = \delta_{\mathcal{K}}(\overline{w}, \overline{z^t})$ for $t > 1$ is not bijective in general. Therefore, if $t > 1$, then the represented output function of $\mathcal{CBPRNG}$ does not necessarily have a full cycle.

4 Simple chain automaton random number generator

In this section, we give an example and then we study the security of our $CBPRNG$ which we call simple chain automaton random number generator (SCARNG). The main advantage of this generator is its simplicity and the fact that it can be easily implemented for IoT devices. The very simple structure of the proposed generator is shown in Fig. 2.

The proposed automaton uses only fundamental arithmetic operations. A detailed pseudocode of SCARNG is shown in Algorithm 1.

Algorithm 1 Simple chain automaton random number generator

```
procedure SCARNG(SIZE, INPUT, AUT, ISTATE)
    for $k = 0 \to SIZE$ do
        for $m = 0 \to 127$ do
            for $i = 0 \to 15$ do
                $STATE[i] \leftarrow ISTATE[i]$
            end for
            $x \leftarrow 0$
            $l \leftarrow 16$
            for $j = 0 \to 15$ do
                $l \leftarrow l - 1$
                if $x = 0$ then
                    $y \leftarrow INPUT[l] + 1$
                    if $y > 255$ then
                        $INPUT[l] \leftarrow 0$
                        $x \leftarrow 0$
                    else
                        $INPUT[l] \leftarrow y$
                        $x \leftarrow 1$
                    end if
                end if
            end for
            for $r = 0 \to ROUND - 1$ do
                for $i = 0 \to 15$ do
                    $a \leftarrow STATE[P[i][15]]$
                    $b \leftarrow INPUT[i]$
                    $x \leftarrow AUT[a][b]$
                    for $j = 15 \to 1$ do
                        $a \leftarrow STATE[P[i][j-1]]$
                        $x \leftarrow AUT[a][x]$
                    end for
                    $STATE[i] \leftarrow x$
                end for
            end for
            for $i = 0 \to 15$ do
                $OARRAY[m][i] \leftarrow STATE[i]$
            end for
        end for
        PRINT($OARRAY$)
    end for
end procedure
```

4.1 Algorithm description

The procedure parameters are the number of random blocks ($SIZE$), the input word ($INPUT$) of the key automaton, the transition matrix of the basic automaton (AUT), and the initial (seed) state of the key automaton ($ISTATE$). Each of the generated random blocks consists of 128 random strings, and each of the random strings is 128 bits long. Thus, the size of the generated random blocks is 2048 byte. The key automaton $\mathcal{K} = (\Sigma^n, \Sigma^{ROUND \times n}, \delta_{\mathcal{K}})$ is a ROUND-component temporal power of an automaton $\mathcal{B} = (\Sigma^n, \Sigma^n, \delta_{\mathcal{B}})$ which is the 16-component chain power of the basic automaton $\mathcal{A} = (\Sigma^n, \Sigma^n, \delta)$ and this basic automaton has a 256×256-type transition matrix which forms a Latin square. Thus, each state and input sign can be represented by a 8-bit binary string. We shall use a 16×16-type auxiliary matrix P, where its first row consists of the vector $0, 1, \ldots, 15$ and all the others are cyclic permutations of the previous one. We will consider $ROUND = 1, 2, 3$ rounds of the output function of $CBPRNG$. The vector INPUT consisting of 16 components represents a single input sign of the key automaton.

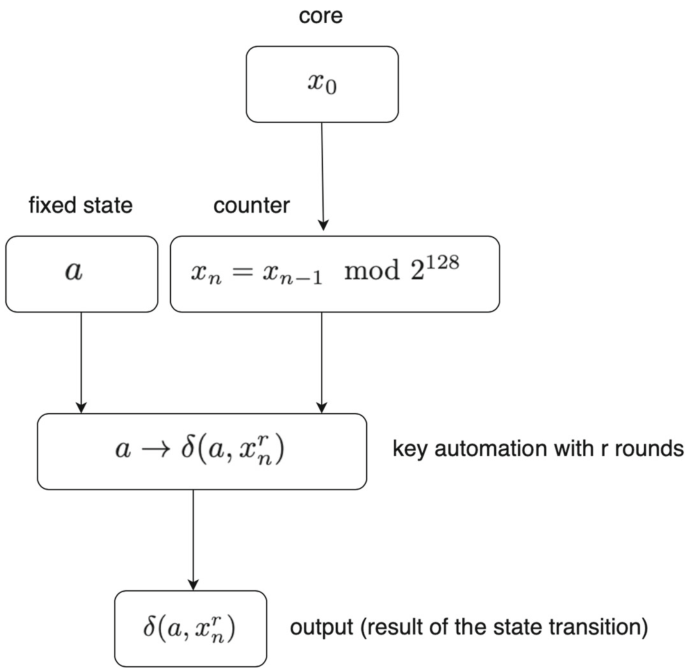
core
x_0
fixed state
counter
a
$x_n = x_{n-1} \mod 2^{128}$
$a \to \delta(a, x_n^r)$
key automation with r rounds
$\delta(a, x_n^r)$
output (result of the state transition)

Fig. 2 Simple chain automaton random number generator (SCARNG) with r rounds

4.2 Toy example

In this section, we show a simple example. Consider the following transition table of an automaton $\mathcal{A} = (\{0, 1, 2, 3\}, \{0, 1, 2, 3\}, \delta)$[3]:

δ	0	1	2	3
0	1	2	3	0
1	3	0	1	2
2	2	3	0	1
3	0	1	2	3

For the sake of simplicity assume that $n = 3$, i.e., the key automaton consists of the three-factor chain power of $\mathcal{A}$, moreover, let the number of rounds be 3. Assume that the subsequent states of the counter, which are the subsequent input signals of the key automaton, are determined by applying the formula $s_{t+1} = s_t + 1(\mod 64)$, or in quaternary number system, $s_{t+1} = s_t + 1(\mod 1000)$, such that the generated pseudorandom number is a three-digit quaternary number. Let the core state s_0, i.e., the initial state of the counter be 203. Therefore, in this case (applying the quaternary number system for calculations), $s_{t+1} = s_t + 1(\mod 1000)$, i.e., the first input signal of the key automaton is

[3] The states of $\mathcal{A}$ are labeled with the elements of the first row, and the input signals are labeled with the elements of the first column of the transition table as usual. Omitting the first row and first column of this table, we receive the transition matrix of $\mathcal{A}$.

Table 1 Parameters used for NIST test suite

Test name	Block length
Block frequency	128
Non-overlapping template	9
Overlapping template	9
Approximate entropy	10
Serial	16
Linear complexity	500

$203+1 = 210$ in the quaternary number system. Let 123 be a fixed state of the key automaton (consisting of the state's first, second, and third component automaton of the chain power of $\mathcal{A}$ represented by the key automaton). In that case, with $a_1a_2a_3 = 123$, $x_1x_2x_3 = 210$, $a_1' = \delta(a_1, \delta(a_2, \delta(a_3, x_1))) = \delta(1, \delta(2, \delta(3, 2))) = \delta(1, \delta(2, 1)) = \delta(1, 1) = 0$, $a_2' = \delta(a_2, \delta(a_3, \delta(a_1', x_2))) = \delta(2, \delta(3, \delta(0, 1))) = \delta(2, \delta(3, 3)) = \delta(2, 3) = 2$, $a;_3 = \delta(a_3, \delta(a_1', \delta(a_2', x_3))) = \delta(3, \delta(0, \delta(2, 0))) = \delta(3, \delta(0, 3)) = \delta(3, 0) = 0$.

Therefore, in the second round we have $a_1a_2a_3 = 020$, and as before, $x_1x_2x_3 = 210$. Thus, we get
$a_1' = \delta(a_1, \delta(a_2, \delta(a_3, x_1))) = \delta(0, \delta(2, \delta(0, 2))) = \delta(0, \delta(2, 2)) = \delta(0, 0) = 1$, $a_2' = \delta(a_2, \delta(a_3, \delta(a_1', x_2))) = \delta(2, \delta(0, \delta(1, 1))) = \delta(2, \delta(0, 0)) = \delta(2, 1) = 1$, $a_3' = \delta(a_3, \delta(a_1', \delta(a_2', x_3))) = \delta(0, \delta(1, \delta(1, 0))) = \delta(0, \delta(1, 2)) = \delta(0, 3) = 0$.

Hence, in the third round we have $a_1a_2a_3 = 110$, and as previously, $x_1x_2x_3 = 210$. Consequently, we get the first pseudorandom number in the following way.
$a_1' = \delta(a_1, \delta(a_2, \delta(a_3, x_1))) = \delta(1, \delta(1, \delta(0, 2))) = \delta(1, \delta(1, 2)) = \delta(1, 3) = 1$, $a_2' = \delta(a_2, \delta(a_3, \delta(a_1', x_2))) = \delta(1, \delta(0, \delta(1, 1))) = \delta(1, \delta(0, 0)) = \delta(1, 1) = 0$, $a_3' = \delta(a_3, \delta(a_1', \delta(a_2', x_3))) = \delta(0, \delta(1, \delta(0, 0))) = \delta(0, \delta(1, 1)) = \delta(0, 0) = 1$.

Then, the first pseudorandom number will be the three-digit quaternary number 101.

Then, we either finish our procedure or generate the second pseudorandom number. When the second pseudorandom number is generated, the procedure is repeated with one and the same state $a_1a_2a_3 = 123$ of the key automaton, and its input signal $s_{t+2} = s_{t+1} + 1(\mod 1000) = 210 + 1(\mod 1000) = 211$ (given in quaternary number system), etc.

5 Experimental results

We implemented Algorithm SCARNG in C++ in order to measure the actual running time and statistical properties of the generator. The test environment was a 2017 MacBook Pro equipped with 7th Generation Kaby Lake 2.9 GHz Intel Core i7 processor (7820HQ) using 16 GB RAM. We have generated 1 GB of random data and applied the NIST statistical randomness test.

5.1 NIST test

The National Institute of Standards and Technology (NIST) published a statistical package consisting of 15 statistical tests that were developed to test the randomness of arbitrarily long binary sequences produced by either hardware- or software-based cryptographic random or pseudorandom number generators [18]. In case of each statistical test, a set of p-values is

Table 2 Results for the uniformity of p-values and the proportion of passing sequences

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	p-value	Proposition	Test name
27	29	29	37	25	41	22	33	31	26	0.361387	297/300	`Frequency`
32	23	32	33	26	41	32	25	35	21	0.257872	294/300	`BlockFrequency`
29	25	38	30	33	23	41	34	21	26	0.175049	299/300	`CumulativeSums`
27	23	23	34	40	33	28	25	33	34	0.389567	299/300	`CumulativeSums`
32	36	24	30	29	24	36	27	29	33	0.779188	300/300	`Runs`
38	26	36	37	30	24	23	24	36	26	0.257872	292/300	`LongestRun`
34	25	39	24	28	21	32	34	28	35	0.372502	294/300	`Rank`
27	39	27	25	19	23	41	37	26	36	0.045675	295/300	`FFT`
38	26	36	32	25	26	32	34	22	29	0.514124	291/300	`NonOverLappingTemp`
30	25	27	31	34	34	44	28	23	24	0.228764	298/300	`NonOverLappingTemp`
												`NonOverLappingTemp`
...	...	...	...	...	...	...	...	...	...	...	...	`NonOverLappingTemp`
30	28	24	27	31	37	42	27	24	30	0.383827	297/300	`OverlappingTemp`
29	31	27	31	37	26	29	21	42	27	0.319084	297/300	`Universal`
24	24	34	32	24	33	30	34	35	30	0.746572	299/300	`ApproximateEntropy`
12	20	20	20	21	16	15	11	21	17	0.600424	173/173	`RandomExcursions`
24	12	14	22	13	24	11	22	16	15	0.120248	172/173	`RandomExcursions`
												`RandomExcursions`
...	...	...	...	...	...	...	...	...	...	...	...	`RandomExcursions`
13	27	20	23	10	13	18	16	17	16	0.142443	172/173	`RandomExcursionsV`
15	22	17	25	17	14	16	20	9	18	0.318190	172/173	`RandomExcursionsV`
												`RandomExcursionsV`
...	...	...	...	...	...	...	...	...	...	...	...	`RandomExcursionsV`
28	26	31	32	29	25	21	35	40	33	0.449672	296/300	`Serial`
23	24	29	32	39	31	33	26	29	34	0.623240	295/300	`Serial`
27	17	32	19	43	26	40	36	24	36	0.005762	296/300	`LinearComplexity`

produced. Given a significance level α, if the p-value is less than or equal to α, then the test suggests that the observed data are inconsistent with our null hypothesis, i.e., the 'hypothesis of randomness,' so we reject it.

A significance level of $\alpha = 0.01$ was used, as it is a standard value for such problems in the field of cryptography and PRNG testing. If the significance level, α, is set to 0.01, it implies that one can expect to reject one sequence out of 100 sequences under the null hypothesis. Therefore, a p-value greater than 0.01 would suggest that the sequence is random, while a p-value less than or equal to 0.01 would indicate that the sequence is non-random.

One of the most important characteristics of a PRNG is the indistinguishability from true random sources. This means that any statistical tests applied to the output of the PRNG should not reveal any computational differences between it and a truly random source. In order to test the quality of SCARNG, the NIST SP-800-22 SP statistical tests were performed using the same parameters as for the AES candidates in order to achieve the most reliable and comparable results. All parameters such as the sequence length, sample size, and significance level were fixed. Namely, these parameters were set to 2^{20} bits, 300 binary sequences, and $\alpha = 0.01$, respectively. The other input parameters are shown in Table 1.

5.2 Minimum number of rounds

Applying only 3 rounds, it was found that the SCARNG algorithm successfully met all of the requirements outlined by the NIST statistical test suite. It has turned out that the NIST statistical test suite is unable to distinguish the output of the algorithm (when using ROUND = 3) from true random sources. Without precise parallelization, the algorithm running time was 27 s to generate 1 GB of random data. The exact p-values of the evaluation of the SCARNG for $ROUND = 3$ are shown in Table 2. We also tested the uniformity of the distribution of the p-values obtained by the statistical tests included in NIST. The uniformity of p-values provided no additional information about the applied PRNG. We have also shown that the proportions of binary sequences which passed the 0.01 level lie in the required confidence interval. For further reading about the testing methodology, we refer to the NIST documentation [18].

Acknowledgements Norbert Tihanyi was supported by the Project no. TKP2021-NVA-29 with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NVA funding scheme.

Funding Open access funding provided by University of Debrecen.

References

1. Coates, R.F.W., Janacek, G.J., Lever, K.V.: Monte Carlo simulation and random number generation. IEEE J. Select. Areas Commun. **6**(1), 58–66 (1988). https://doi.org/10.1109/49.192730
2. Tihanyi, N., Borsos, B.: A general construction for generating pseudorandom sequences using the digit expansion of real functions. In: 2020 22nd International Symposium on Symbolic and Numeric Algorithms

for Scientific Computing (SYNASC), pp. 48–54. IEEE, Timisoara, Romania (2020). https://doi.org/10.1109/SYNASC51798.2020.00019. https://www.ieeexplore.ieee.org/document/9357086/. Accessed 2023-04-20

3. Stojanovski, T., Pihl, J., Kocarev, L.: Chaos-based random number generators. Part II: practical realization. IEEE Trans. Circuits Syst.I: Fundam. Theory Appl. **48**(3), 382–385 (2001). https://doi.org/10.1109/81.915396
4. Wolfram, S.: Random sequence generation by cellular automata. Adv. Appl. Math. **7**(2), 123–169 (1986). https://doi.org/10.1016/0196-8858(86)90028-X
5. Hortensius, P.D., McLeod, R.D., Card, H.C.: Parallel random number generation for VLSI systems using cellular automata. IEEE Trans. Comput. **38**(10), 1466–1473 (1989). https://doi.org/10.1109/12.35843
6. Hortensius, P.D., McLeod, R.D., Pries, W., Miller, D.M., Card, H.C.: Cellular automata-based pseudo-random number generators for built-in self-test. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **8**(8), 842–859 (1989). https://doi.org/10.1109/43.31545
7. Guan, S.-U., Zhang, S.: Pseudorandom number generation based on controllable cellular automata. Futur. Gener. Comput. Syst. **20**(4), 627–641 (2004). https://doi.org/10.1016/S0167-739X(03)00128-6
8. Guan, S.-U., Tan, S.K.: Pseudorandom number generation with self-programmable cellular automata. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **23**(7), 1095–1101 (2004). https://doi.org/10.1109/TCAD.2004.829808
9. Comer, J.M., Cerda, J.C., Martinez, C.D., Hoe, D.H.K.: Random number generators using Cellular Automata implemented on FPGAs. In: Proceedings of the 2012 44th Southeastern Symposium on System Theory (SSST), pp. 67–72. IEEE, Jacksonville, FL, USA (2012). https://doi.org/10.1109/SSST.2012.6195137 . https://www.ieeexplore.ieee.org/document/6195137/. Accessed 2023-04-01
10. Tomassini, M., Perrenoud, M.: Cryptography with cellular automata. Appl. Soft Comput. **1**(2), 151–160 (2001). https://doi.org/10.1016/S1568-4946(01)00015-1
11. Bao, F.: Cryptanalysis of a partially known cellular automata cryptosystem. IEEE Trans. Comput. **53**(11), 1493–1497 (2004). https://doi.org/10.1109/TC.2004.94
12. Zhang, T.J., Manhrawy, I.M., Abdo, A.A., El-Latif, A.A.A., Rhouma, R.: Cryptanalysis of elementary cellular automata based image encryption. Adv. Mater. Res. **981**, 372–375 (2014). https://doi.org/10.4028/www.scientific.net/AMR.981.372
13. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010). https://doi.org/10.1016/j.comnet.2010.05.010
14. Stellios, I., Kotzanikolaou, P., Psarakis, M., Alcaraz, C., Lopez, J.: A survey of IoT-enabled cyberattacks: assessing attack paths to critical infrastructures and services. IEEE Commun. Surv. Tutor. **20**(4), 3453–3495 (2018). https://doi.org/10.1109/COMST.2018.2855563
15. Dömösi, P., Gáll, J., Horváth, G., Tihanyi, N.: Statistical analysis of DH1 cryptosystem. Acta Cybernet. **23**(1), 371–378 (2017). https://doi.org/10.14232/actacyb.23.1.2017.20
16. Dömösi, P., Gáll, J., Horváth, G., Tihanyi, N.: Some remarks and tests on the DH1 cryptosystem based on automata compositions. Informatica (2019). https://doi.org/10.31449/inf.v43i2.2687
17. Borsos, B., Dömösi, P., Alhammadi, Y., Tihanyi, N., Gáll, J., Horváth, G.: A pseudorandom number generator with full cycle length based on automata compositions. Informatica (2021). https://doi.org/10.31449/inf.v45i2.3109
18. Bassham, L.E., Rukhin, A.L., Soto, J., Nechvatal, J.R., Smid, M.E., Barker, E.B., Leigh, S.D., Levenson, M., Vangel, M., Banks, D.L., Heckert, N.A., Dray, J.F., Vo, S.: A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical Report NIST SP 800-22r1a, National Institute of Standards and Technology, Gaithersburg, MD (2010). https://doi.org/10.6028/NIST.SP.800-22r1a. Edition: 0. https://www.nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf. Accessed 2023-04-17
19. Dömösi, P., Nehaniv, C.L.: Algebraic Theory of Automata Networks: an Introduction. SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics, Philadelphia (2005)
20. Salmon, J.K., Moraes, M.A., Dror, R.O., Shaw, D.E.: Parallel random numbers: as easy as 1, 2, 3. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–12. ACM, Seattle Washington (2011). https://doi.org/10.1145/2063384.2063405