A Full Cycle Length Pseudorandom Number Generator Based on Compositions of Automata

Pál Dömösi
Institute of Mathematics and Informatics, University of Nyíregyháza, H-4400 Nyíregyháza, Sóstói út 31/B, Hungary
Faculty of Informatics, University of Debrecen, H-4028 Debrecen, Kassai út 26, Hungary
E-mail: domosi@unideb.hu, domosi.pal@nye.hu

József Gáll and Géza Horváth
Faculty of Informatics, University of Debrecen, H-4028 Debrecen, Kassai út 26, Hungary
E-mail: gall.jozsef@inf.unideb.hu, horvath.geza@inf.unideb.hu

Bertalan Borsos
Faculty of Informatics, Eötvös Loránd University, H-1117 Budapest, Pázmány Péter sétány 1/C, Hungary
E-mail: bertalanborsos@gmail.com

Norbert Tihanyi
Digital14 LLC, xen1thLabs, Cryptography Laboratory, Abu Dhabi, United Arab Emirates
E-mail: norbert.tihanyi@digital14.com

Yousef Al Hammadi
College of Information Technology, United Arab Emirates University
P.O. Box 15551, Al Ain, Abu Dhabi, United Arab Emirates
E-mail: yousef-A@uaeu.ac.ae

Keywords: automata network, NIST test, block cipher, pseudo random number generator, composition of automata, Gluškov product of automata, temporal product of automata

Received: April 3, 2020

In this paper a new Pseudorandom Number Generator based on compositions of abstract automata is presented. We show that it has full cycle with length of 2^{128}. It is also shown that the output satisfies the statistical requirements of the NIST randomness test suite.

Povzetek: V prispevku je predstavljen nov psevdonaključni generator števil.

1 Introduction

In this paper, the authors continue their joint research of cryptographic tools based on compositions of abstract finite automata started in [6].

Random number generators have been used for a wide variety of fields and purposes, such as cryptography, pattern recognition, gambling and VLSI testing. [18]. In this paper a Pseudorandom Number Generator (PRNG) based on automata theory will be introduced and studied. The most frequent type of automata theory based pseudorandom generators are implemented on the basis of cellular automata. The first pseudorandom number generator based on cellular automata was proposed by S. Wolfram [29, 30] and there are many pseudorandom generators based on cellular automata today. (See [2]-[5], [8]-[16], [19, 20],[23]-[28]).

A common problem of some well-known pseudorandom generators based on cellular automata is that they have serious application difficulties: some of them can be broken [1],[17], while in case of others the selection of the key automaton poses difficulties [10].

These reasons, among others, justify the use of new automata theory-based pseudorandom number generators based on principles other than cellular automata.

Counter-based random number generation [22] is a relatively new technique for creating a pseudorandom number generator using only an integer counter as the internal state of the generator. The state transition function is an increment by one modulo the size n of the finite state set $S = \{0, \ldots, n-1\}$ and the complexity comes in the map from the state to the random sample. Formally, a counter based random number generator (CBRNG) is a structure $\mathcal{CBRNG} = (K, Z_J, S, f, U, g)$, 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. If Z_J 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 $\mathcal{CBRNG}$ has a *simple output multiplicity*. 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 k-times round of $g : K \times S \to U$ for some $k > 2$ 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 $k-1$-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.

The $\mathcal{CBRNG} = (K, Z_J, S, f, U, g)$ 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 $g(k,0,f^1(s)), \ldots, g(k,J-1,f^1(s))$, $g(k,0,f^2(s)), \ldots, g(k,J-1,f^2(s)), g(k,0,f^n(s))$, $, \ldots, g(k,J-1,f^n(s))$, 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 $(g(k,0,f(s)), \ldots g(k,J-1,f(s))$ is called the output vector of initial state.

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

The following statement is clear.

***Proposition* 1** A $\mathcal{CBRNG} = (K, Z_J, S, f, U, g)$ 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 $g_k(s) = (g(k,0,s), \ldots, g(J-1,s)), s \in S$ is bijective.

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

2 Preliminaries

We start with some standard concepts and notation. For all notions and notation not defined here we refer to the monograph [7]. By an *automaton* we mean a deterministic finite automaton without outputs. In more detail, 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* $\delta : A \times \Sigma \to A$. The transition matrix of an automaton is a matrix with rows corresponding to each input and columns corresponding to each state; at the entry of any row indicated by an input $x \in \Sigma$ sign and any column indicated by a state $a \in A$ the state $\delta(a,x)$ is put. If all rows of the transition matrix are permutations of the state set then we speak about *permutation automaton*.

A Latin square of order n is an $n \times n$ matrix (with n rows and n columns) in which the elements of an n-state set $\{a_0, a_1, \ldots, a_{n-1}\}$ are entered so that each element occurs exactly once in each fixed (row, column) pair.

In this paper we will consider special compositions of automata consisting of component automata such that all components have the same transition matrix of the Latin square form. We will show that these compositions of automata are permutation automata, moreover for every state of these automata compositions it has a very low likelihood that two randomly chosen distinct input signs take the automaton into the same state. By these properties, we would like to avoid vulnerability to statistical attacks. Finally we note that, apart from the trivial cases, the transition matrices of the considered automata compositions are not quadratic. Therefore, their transition matrix can not form Latin squares.

3 Construction

We start with some standard definitions. (See, for example [6, 7]).

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* $\varphi_i : A_1 \times \cdots \times A_n \times \Sigma \to \Sigma_i$ 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.*

Given a function $f : X_1 \times \cdots \times X_n \to Y$, we say that f is *really independent of its i-th variable* if for every pair $(x_1, \ldots, x_n), (x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots, x_n) \in X_1 \times \cdots \times X_n$, $f(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots, x_n)$. Otherwise we say that f *really depends on its i-th variable*. A (finite) *directed graph* (or, in short, a *digraph*) $\mathcal{D} = (V, E)$ (of order $n > 0$) is a pair consisting of sets of *vertices* $V = \{v_1, \ldots, v_n\}$ and *edges* $E \subseteq V \times V$. Elements of V are sometimes called *nodes*. If $|V| = n$ then we also say that $\mathcal{D}$ is a digraph of order n. $\mathcal{D}$ is called *bipartite* if its vertices can be partioned into two sets A, B such that every (direct) edge connects a vertex in A to a vertex in B or vica versa. Further on, we will assume that V is an ordered set of integers $1, \ldots n$ for some positive integer n. Given a digraph $\mathcal{D} = (V, E)$, we say that the above defined Gluškov product is a $\mathcal{D}$-product if for every pair $i, j \in \{1, \ldots, n\}$, $(i,j) \notin E$ implies that the feedback function φ_i is really independent of its j-th variable. Let Σ be the set of all ℓ (preferably 4 or 8) length binary strings for a given length $\ell > 0$. Moreover, let $\mathcal{A}_i = (\Sigma, \Sigma, \delta_{\mathcal{A}_i}), i = 2, \ldots, n$ be copies of $\mathcal{A}_1$, and let n be a positive integer power of 2. Consider the following

simple bipartite digraphs:
$\mathcal{D}_1 = (\{1,\ldots,n\},\{(n/2+1,1),(n/2+2,2),\ldots,(n,n/2)\})$,
$\mathcal{D}_2 = (\{1,\ldots,n\},\{(n/4+1,1),(n/4+2,2),\ldots,(n/2,n/4),$
$(3n/4+1,n/2+1),(3n/4+2,n/2+2),\ldots,(n,3n/4)\})$,
...,
$\mathcal{D}_{\log_2 n-1} = (\{1,\ldots,n\},\{(3,1),(4,2),(7,5),(8,6),\ldots,(n-1,n-3),(n,n-2)\})$,
$\mathcal{D}_{\log_2 n} = (\{1,\ldots,n\},\{(2,1),(4,3),\ldots,(n,n-1)\})$,
$\mathcal{D}_{\log_2 n+1} = \mathcal{D}_1$.

For every digraph $\mathcal{D} = (V,E)$ with $\mathcal{D} \in \{\mathcal{D}_1,\ldots,\mathcal{D}_{\log_2 n}\}$, let us define the Gluškov-type power, called *two-phase $\mathcal{D}$-power*, $\mathcal{A}_\mathcal{D} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n(\Sigma^n,(\varphi_1,\ldots,\varphi_n))$ of $\mathcal{A}_1$ (with $\mathcal{A}_1 = \mathcal{A}_2 \ldots = \mathcal{A}_n$) so that for every $(a_1,\ldots,a_n),(x_1,\ldots,x_n) \in \Sigma^n, (i,j) \in E$, $\varphi_i(a_1,\ldots,a_n,(x_1,\ldots,x_n)) = a'_j \oplus x_j$, and $\varphi_j(a_1,\ldots,a_n,(x_1,\ldots,x_n)) = a_i \oplus x_i$, where $a_i \oplus x_i$ is the bitwise addition modulo 2 of a_i and x_i, a'_j denotes the state into which $\varphi_j(a_1,\ldots,a_n,(x_1,\ldots,x_n))$ takes the automaton $\mathcal{A}_i$ from its state a_j, and $a'_j \oplus x_j$ is the bitwise addition modulo 2 of a'_j and x_j. [1]

Next we define the concept of *temporal product* of automata. Let $\mathcal{A}_t = (A,\Sigma_t,\delta_t), t = 1,2$ be automata having a common state set A. Take a finite nonvoid set Σ and a mapping φ of Σ into $\Sigma_1 \times \Sigma_2$. Then the automaton $\mathcal{A} = (A,\Sigma,\delta)$ is a *temporal product* (t-product) of $\mathcal{A}_1$ by $\mathcal{A}_2$ with respect to Σ and φ if for any $a \in A$ and $x \in \Sigma$, $\delta(a,x) = \delta_2(\delta_1(a,x_1),x_2)$, where $(x_1,x_2) = \varphi(x)$. The concept of temporal product is generalized in the natural way to an arbitrary finite family of $n > 0$ automata $\mathcal{A}_t$ $(t = 1,\ldots,n)$, all with the same state set A.

***Proposition* 2** Suppose that $\mathcal{A}_1 = (\Sigma,\Sigma,\delta)$ is a permutation automaton. Then for every $i = 1,\ldots,\log_2 n$, the $\mathcal{D}_i$-power of $\mathcal{A}_1$ also forms a permutation automaton.

Proof. Assume that $\mathcal{A}_1$ 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 $a,b \in A_1$ and input $x \in \Sigma$, $\delta_1(a,x) = \delta_1(b,x)$ implies $a = b$.

By our above observation, if the $\mathcal{D}_i$-power $\mathcal{A}_{\mathcal{D}_i} = (\Sigma^n,\Sigma^n,\delta_{\mathcal{D}_i})$ is a permutation automaton, then for every pair of states $(a_1,\ldots,a_n),(b_1,\ldots,b_n)$ and input sign $(x_1,\ldots,x_n)$, we have $\delta_{\mathcal{D}_i}((a_1,\ldots,a_n),(x_1,\ldots,x_n)) = \delta_{\mathcal{D}_i}((b_1,\ldots,b_n),(x_1,\ldots,x_n))$ that implies $(a_1,\ldots,a_n) = (b_1,\ldots,b_n)$.

Suppose that $\mathcal{A}_{\mathcal{D}_i}$ is not a permutation automaton. Then, by our observations, there are a pair of distinct states $(a_1,\ldots,a_n),(b_1,\ldots,b_n)$ and an input sign $(x_1,\ldots,x_n)$ for which $\delta_{\mathcal{D}_i}((a_1,\ldots,a_n),(x_1,\ldots,x_n)) = \delta_{\mathcal{D}_i}((b_1,\ldots,b_n),(x_1,\ldots,x_n))$.

If $(a_1,\ldots,a_n)$ and $(b_1,\ldots,b_n)$ are distinct then there exists an index $j \in \{1,\ldots,n\}$ with $a_j \neq b_j$. Put $(a'_1,\ldots,a'_n) = \delta_{\mathcal{D}_i}((a_1,\ldots,a_n),(x_1,\ldots,x_n))$ and $(b'_1,\ldots,b'_n) = \delta_{\mathcal{D}_i}((b_1,\ldots,b_n),(x_1,\ldots,x_n))$. It is enough to show that, in this case, $(a'_1,\ldots,a'_n) \neq (b'_1,\ldots,b'_n)$.

Let E denote the set of edges in $\mathcal{D}_i$.

First we suppose $(j,k) \in E$ for some $k \in \{1,\ldots,n\} \setminus \{j\}$. Then, by definition, $a'_j = \delta_{\mathcal{A}_1}(a_j,a'_k \oplus x_k)$ and $b'_j = \delta_{\mathcal{A}_1}(b_j,b'_k \oplus x_k)$. Because $\mathcal{A}_1$ is a permutation automaton, by the assumption $a'_k = b'_k$, we get $a_j = b_j$, a contradiction. Therefore $a'_k \neq b'_k$. Hence, $(a'_1,\ldots,a'_n) \neq (b'_1,\ldots,b'_n)$.

Next we assume $(k,j) \in E$ for some $j \in \{1,\ldots,n\} \setminus \{k\}$. Obviously, by $a'_j \neq b'_j$ we have $(a'_1,\ldots,a'_n) \neq (b'_1,\ldots,b'_n)$ and then we are done once again. Therefore, suppose $a'_j = b'_j$. Then, again by definition, $a'_j = \delta_{\mathcal{A}_1}(a_j,a_k \oplus x_k)$ and $b'_j = \delta_{\mathcal{A}_1}(b_j,b_k \oplus x_k)$. Because $\mathcal{A}_1$ is a permutation automaton and $a_k \neq b_k$, $a'_j = b'_j$ is possible only if $a_j = b_j$, a contradiction. Therefore, $a'_j \neq b'_j$ and thus we obtain $(a'_1,\ldots,a'_n) \neq (b'_1,\ldots,b'_n)$. The proof is complete. QED

Now we give an alternative proof of Lemma 2 in [6].

***Proposition* 3** Temporal products of permutation automata are also permutation automata.

Proof. Obviously, it is enough to prove our statement for temporal products of two components. Thus, let $\mathcal{M} = (M,\Sigma,\delta_\mathcal{M})$ be a temporal product of permutation automata $\mathcal{M}_1 = (M,\Sigma_1,\delta_{\mathcal{M}_1})$ and $\mathcal{M}_2 = (M,\Sigma_2,\delta_{\mathcal{M}_2})$ (having the same state set) with respect to Σ and $\varphi : \Sigma \to \Sigma_1 \times \Sigma_2$. Let $m_1,m_2 \in M$ be distinct states and $x \in \Sigma$ be an arbitrary input sign of $\mathcal{M}$. Moreover, let $\varphi(x) = (x_1,x_2)$ for some $x_1 \in \Sigma_1$ and $x_2 \in \Sigma_2$. Then for every distinct pair $m_1,m_2 \in M$ of states and $x_1 \in \Sigma_1$ we have $\delta_{\mathcal{M}_1}(m_1,x_1) \neq \delta_{\mathcal{M}_2}(m_2,x_2)$. On the other hand, $\mathcal{M}_2$ is also a permutation automaton. Therefore, because of $\delta_{\mathcal{M}_1}(m_1,x) \neq \delta_{\mathcal{M}_2}(m_2,x)$, for every $x_2 \in \Sigma_2$, $\delta_{\mathcal{M}_1}(\delta_{\mathcal{M}_1}(m_1,x_1),x_2) \neq \delta_{\mathcal{M}_2}(\delta_{\mathcal{M}_2}(m_2,x_1),x_2)$. Thus, using $\varphi(x) = (x_1,x_2), \delta_\mathcal{M}(m_1,x) \neq \delta_\mathcal{M}(m_2,x)$. In other words, for every distinct pair $m_1,m_2 \in M$ of states and input sign $x \in \Sigma, \delta_\mathcal{M}(m_1,x) \neq \delta_\mathcal{M}(m_2,x)$. But then all rows of the transition matrix of $\mathcal{M}$ are permutations of the state set. This completes the proof. QED

Let $\mathcal{B} = (\Sigma^n,(\Sigma^n)^{\log_2 n},\delta_\mathcal{B})$ be the temporal product of $\mathcal{A}_{\mathcal{D}_1},\ldots,\mathcal{A}_{\mathcal{D}_{\log_2 n}}$ with respect to $(\Sigma^n)^{\log_2 n}$ and the identity map $\varphi : (\Sigma^n)^{\log_2 n} \to (\Sigma^n)^{\log_2 n}$, where $\mathcal{A}_{\mathcal{D}_i}, i = 1,\ldots,\log_2 n$ is a $\mathcal{D}_i$ - power of the automaton $\mathcal{A}_1 = (\Sigma,\Sigma,\delta_{\mathcal{A}_1})$.

From now on we assume that $\mathcal{A}_1$ is a permutation automaton having $\delta_{\mathcal{A}_1}(a,x) \neq \delta_{\mathcal{A}_1}(a,x')$ for every $a,x,x' \in \Sigma, x \neq x'$, and we say that $\mathcal{B}$ is a *key-automaton* with respect to the permutation automaton $\mathcal{A}_1$ called the basic automaton of $\mathcal{B}$.[2]

Theorem 1 in [6] concerns key automata consisting of basic automata having a transition table forming a Latin

[1] We remark that there are $V_1,V_2 \subset V$ with $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$ so that for every $j \in V_2$ there exists exactly one $i \in V_1$ with $(j,i) \in E$. Therefore all the functions $\varphi_1,\ldots,\varphi_n$ are well-defined.

[2] Recall that n should be a positive integer power of 2.

cube. The next statement is formally the same but, of course, it concerns key automata consisting of basic automata having a transition table forming a Latin square. By this fact, the next statement could also be derived from Theorem [6] using some simplification regarding its proof.

We note that the following statement is formally the same as Theorem 1 [6] which is concerning key automata consisting of basic automata having a little bit different structure as basic automata of the present paper. (The transition table of basic automata in [6] forms a Latin cube while the transition table of basic automata of the present paper forms a Latin square.) This fact implies that the automata compositions discussed in the present paper are more or less similar to the ones in [6].

For the sake of simplicity, we give a direct proof of the next statement which, using some simplifications, can also be derived from the proof of Theorem 1 in [6] .

***Theorem* 4** Every key automaton is a permutation automaton.

Proof. Consider a key automaton $\mathcal{B} = (\Sigma^n, (\Sigma^n)^{\log_2 n}, \delta_{\mathcal{B}})$ and its basic automaton $\mathcal{A}_1 = (A_1, \Sigma_1, \delta 1)$. By the definition of key automaton, $\mathcal{A}_1$ is a permutation automaton.

Therefore, using Proposition 2, for every permutation automaton $\mathcal{A}_1$ and $i = 1, 2, \ldots, \log_2 n$, the $\mathcal{D}_i$-power $\mathcal{A}_{\mathcal{D}_i} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n(\Sigma^n, (\varphi_1, \ldots, \varphi_n))$ of $\mathcal{A}_1$ is a permutation automaton.

Recall that $\mathcal{B}$ is a temporal product of $\mathcal{A}_{\mathcal{D}_1}, \ldots, \mathcal{A}_{\mathcal{D}_{\log_2 n}}$. Therefore, by Proposition 3, our proof is done. QED

***Proposition* 5** Suppose that the transition matrix of an automaton $\mathcal{A}_1 = (\Sigma, \Sigma, \delta)$ forms a Latin square. Then for every $i = 1, \ldots, \log_2 n$, the transition matrix of the $\mathcal{D}_i$-power of $\mathcal{A}_1$ also forms a Latin square.

Proof. Obviously, if the transition matrix of an automaton $\mathcal{A}_i = (Ai, \Sigma, \delta)$ forms a Latin square then $\mathcal{A}$ is a permutation automaton. But then, by Proposition 2, all of $\mathcal{D}_i$-powers are permutation automata. In other words, the rows of their transition matrices form a permutation of their state set. All that is left is to show that the columns of their transition matrices also have this property.

Consider a $\mathcal{D}_i$-power $\mathcal{A}_{\mathcal{D}_i} = (\Sigma^n, \Sigma^n, \delta_{\mathcal{D}_i})$ of $\mathcal{A}_1$ having $\mathcal{A}_{\mathcal{D}_i} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n(\Sigma^n, (\varphi_1, \ldots, \varphi_n))$ for some $i = 1, 2, \ldots, \log_2 n$. We should prove that for every state $(a_1, \ldots, a_n) \in \Sigma^n$ and distinct words $(x_1, \ldots, x_n), (y_1, \ldots, y_n) \in \Sigma^n$, $\delta_{\mathcal{D}_i}((a_1, \ldots, a_n), (x_1, \ldots, x_n)) \neq \delta_{\mathcal{D}_i}((a_1, \ldots, a_n), (y_1, \ldots, y_n))$. Let $j \in \{1, \ldots, n\}$ be an index with $x_j \neq y_j$. Moreover, let E be the set of edges in $\mathcal{D}_i$ and let us assume that $(i, j) \in E$ (with $i \in \{1, \ldots, n\} \setminus \{j\}$). Then $x_i \neq y_i$ implies $\delta(a_j, a_i \oplus x_i) \neq \delta(a_j, a_i \oplus y_i)$. Therefore, by our construction, the j-th components of $\delta_{\mathcal{D}_i}((a_1, \ldots, a_n), (x_1, \ldots, x_n))$ and $\delta_{\mathcal{D}_i}((a_1, \ldots, a_n), (y_1, \ldots, y_n))$ are distinct as we stated. Now we assume $(i, j) \in E$ (with $i \in \{1, \ldots, n\} \setminus \{j\}$). If $\delta(a_i, a_j \oplus x_j) \neq \delta(a_i, a_j \oplus y_j)$ then the i-th components of $\delta_{\mathcal{D}_i}((a_1, \ldots, a_n), (x_1, \ldots, x_n))$ and $\delta_{\mathcal{D}_i}((a_1, \ldots, a_n), (y_1, \ldots, y_n))$ are distinct again. Therefore, let $\delta(a_i, a_j \oplus x_j) = \delta(a_i, a_j \oplus y_j)$. But then, by $x_i \neq y_i$, we receive $\delta(a_i, a_j \oplus x_j) \oplus x_i \neq \delta(a_i, a_j \oplus y_j) \oplus y_i$. Obviously, this leads to $\delta(a_j, \delta(a_i, a_j \oplus x_j) \oplus x_i) \neq \delta(a_j, \delta(a_i, a_j \oplus y_j) \oplus y_i)$. Thus we get again that the j-th component of $\delta_{\mathcal{D}_i}((a_1, \ldots, a_n), (x_1, \ldots, x_n))$ and $\delta_{\mathcal{D}_i}((a_1, \ldots, a_n), (y_1, \ldots, y_n))$ are distinct. This finishes the proof. QED

Obviously, if the automaton $\mathcal{A}_1$ has a transition table forming a Latin square then it is a permutation automaton. Therefore, it can be a basic automaton of an appropriate key automaton.

Observation 6 Consider a key automaton $\mathcal{B}$ for which its basic automaton $\mathcal{A}_1$ has a transition table forming a Latin square. Then for every state a of $\mathcal{B}$, the probability that its two random signs take $\mathcal{B}$ from a into the same state is $((2|\Sigma| - 1)/|\Sigma|^3)^{n/2}$, where Σ is the set of states of $\mathcal{A}_1$ and n is the number of (identical) component automata in each $\mathcal{D}_i$ power ($i = 1, \ldots, \log_2 n$) for which $\mathcal{B}$ is a temporal product of these $\mathcal{D}_i$ powers.

Proof. Denote by $\mathcal{M} = (\Sigma^n, (\Sigma^n)^{\log_2 n - 1}, \delta_{\mathcal{M}})$ the temporal product consisting of the first $\log_2 n - 1$ components $\mathcal{A}_{\mathcal{D}_i}, i = 1, \ldots, \log_2 n - 1$ of the key automaton $\mathcal{B} = (\Sigma^n, (\Sigma^n)^{\log_2 n}, \delta_{\mathcal{B}})$ and consider the $\log_2 n$-th component $\mathcal{A}_{\mathcal{D}_{\log_2 n}} = (\Sigma^n, \Sigma^n, \delta_{\mathcal{D}_{\log_2 n}})$ of $\mathcal{B}$. By Proposition 2 and Proposition 3, $\mathcal{M}$ is a permutation automaton. Therefore, for every state $(a_1, \ldots, a_n) \in \Sigma^n$ of $\mathcal{M}$ its random input signs $w \in (\Sigma^n)^{\log_2 n - 1}$ take $\mathcal{M}$ into each state with the same probabbiliy $1/|\Sigma^n|$.

Consider a fixed state $(c_1, \ldots, c_n) \in \Sigma^n$ and a randomly chosen pair $w_1, w_2 \in (\Sigma^n)^{\log_2 n - 1}$ of $\mathcal{M}$ and denote by $(a_1, \ldots, a_n), (a_1, \ldots, a_n) \in M$ the pair of states in $\mathcal{M}$ such that $\delta_{\mathcal{M}}((c_1, \ldots, c_n), w_1) = (a_1, \ldots, a_n)$ and $\delta_{\mathcal{M}}((c_1, \ldots, c_n), w_2) = (a_1, \ldots, a_n)$.

Moreover consider a randomly chosen pair $(x_1, \ldots, x_n), (y_1, \ldots, y_n) \in \Sigma^n$ of input signs of $\mathcal{A}_{\mathcal{D}_{\log_2 n}}$, and assume that $\delta_{\mathcal{D}_{\log_2 n}}((a_1, \ldots, a_n), (x_1, \ldots, x_n)) = \delta_{\mathcal{D}_{\log_2 n}}(b_1, \ldots, b_n), (y_1, \ldots, y_n))$. For every $i = 1, \ldots, n$, there exists a single $i \in \{1, \ldots, n\}$ such that either $(i, j) \in E$ or $(j, i) \in E$, where E denotes the set of edges in the digraph $\mathcal{D}_{\log_2 n}$. There are the following cases. If $(i, j) \in E$ then $\delta(a_i, \delta(a_j, a_i \oplus x_i) \oplus x_j)$ and $\delta(b_i, \delta(b_j, b_i \oplus y_i) \oplus y_j))$ will be the i-th and $\delta(a_j, a_i \oplus x_i)$ and $\delta(b_j, b_i \oplus y_i)$ will be the j-th component of $\delta_{\mathcal{D}_{\log_2 n}}((a_1, \ldots, a_n), (x_1, \ldots, x_n))$ and $\delta_{\mathcal{D}_{\log_2 n}}(b_1, \ldots, b_n), (y_1, \ldots, y_n))$, respectively. By the assumption $\delta_{\mathcal{D}_{\log_2 n}}((a_1, \ldots, a_n), (x_1, \ldots, x_n)) = \delta_{\mathcal{D}_{\log_2 n}}(b_1, \ldots, b_n), (y_1, \ldots, y_n))$, we have $\delta(a_i, \delta(a_j, a_i \oplus x_i) \oplus x_j) = \delta(b_i, \delta(b_j, b_i \oplus y_i) \oplus y_j)$ and $\delta(a_j, a_i \oplus x_i) = \delta(b_j, b_i \oplus y_i)$ By the second equality, we can write the first one in the form $\delta(a_i, \delta(a_j, a_i \oplus x_i) \oplus x_j) = \delta(b_i, \delta(a_j, a_i \oplus x_i) \oplus y_j)$. Recall that the transition matrix of $\mathcal{A}_1$ forms a Latin square. Therefore, by these considered equalities, $a_i = b_i$ if and only if $x_j = y_j$ and $a_j = b_j$ if and only if $x_i = y_i$.

On the other hand, for every $k = 1, \ldots, n$, there are $|\Sigma|^2$

cases having $a_k = b_k$ and $x_k = y_k$, $a_k, b_k, x_k, y_k \in \Sigma$. Of course, all of these cases take the i-th and j-th components of $\mathcal{A}_{\mathcal{D}_{\log_2 n}}$ into the same state.

In addition, every element of Σ appears exactly $|\Sigma|$-times in the transition table of $\mathcal{A}_1$ because it forms a Latin square. Moreover, we can consider only nonequal pairs of quadruplets.

Hence there are $|\Sigma|(|\Sigma|-1)$ number of quadruple possibilities $a_i, b_i, x_i, y_i \in \Sigma, i = 1, \ldots, n$ having $a_i \neq b_i, x_i \neq y_i$ taking the i-th and j-th components of $\mathcal{A}_{\mathcal{D}_{\log_2 n}}$ into the same state.

In sum, we have that the probability that $\delta(a, x)$ and $\delta(b, y)$ coincide for a random quadruple $a, b, x, y \in \Sigma$ is $(2|\Sigma|^2 - |\Sigma|)/|\Sigma|^4 = (2|\Sigma| - 1)/|\Sigma|^3$.

By our constructions, the digraph $\mathcal{D}_{\log_2 n}$ has $n/2$ edges. Consequently, the probability that a random quadruple $(a_1, \ldots, a_n), (b_1, \ldots, b_n), (x_1, \ldots, x_n), (y_1, \ldots, y_n) \in \Sigma^n$ has the property $\delta_{\mathcal{D}_{\log_2 n}}((a_1, \ldots, a_n), (x_1, \ldots, x_n)) = \delta_{\mathcal{D}_{log_2 n}}(b_1, \ldots, b_n), (y_1, \ldots, y_n))$ is $((2|\Sigma| - 1)/|\Sigma|^3)^{n/2}$.

We remark that if we have an implementation with $|\Sigma| = 256$ and $n = 16$ then the considered probability is $((512 - 1)/256^3)^8 \approx 1/2^{120}$.

By our investigations, we receive that the probability of the equality $\delta_{\mathcal{B}}((c_1, \ldots, c_n), w_1(x_1, \ldots, x_n)) = \delta_{\mathcal{B}}((c_1, \ldots, c_n), w_2(y_1, \ldots, x_n))$ is $((2|\Sigma| - 1)/|\Sigma|^3)^{n/2}$, whenever $w_1(x_1, \ldots, x_n)$ and $w_2(y_1, \ldots, x_n)$ are randomly chosen input signs of $\mathcal{B}$. QED

***Theorem* 7** Every key automaton transition function can be applied as an output function of a counter-based pseudo random number generator.

Proof. As the proof of our statement, we give a construction of an appropriate counter based PRNG (CBRNG) $\mathcal{CBRNG} = (K, Z_J, S, f, U, g)$ 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). For sake of simplicity, assume that $\mathcal{CBRNG}$ has a simple output multiplicity, i.e., let $Z_J = \{0\}$ be a singleton (although it may have more than one element). Thus the state space is $S = \{0, \ldots, m-1\}$. The elements of S may be considered binary strings of fixed length. Therefore, we may assume that S coincides with the state set Σ^n of the key automaton. We assume $K \subseteq S \times (\Sigma^n)^{2\log_2 n}$, where $S = \Sigma^n$ is the state set, and $(\Sigma^n)^{2\log_2 n}$ is the input set of the key automaton $\mathcal{B} = (\Sigma^n, (\Sigma^n)^{\log_2 n}, \delta_{\mathcal{B}})$. The first component of the elements of K are considered as possible seeds of the random number generator and the second one is an input element of $\mathcal{B}$. The output space U and the state space S coincide. The output function $g : K \times S \to U$ is given as $g(k, (a_1, \ldots, a_n)) = b_1||b_2|| \ldots ||b_n, k \in K, (a_1, \ldots, a_n) \in S(= \Sigma^n)$, where $b_1||b_2|| \ldots ||b_n$ is the concatenation of $b_1, \ldots, b_n$ as binary strings and $(b_1, \ldots, b_n)$ is a state of the key automaton $\mathcal{B}$ with $(b_1, \ldots, b_n) = \delta_{\mathcal{B}}((a_1, \ldots, a_n), (x_1, \ldots, x_n))$, where the concatenation $a_1|| \ldots ||a_n$ of $a_1, \ldots, a_n$ as binary strings is given by $a_1|| \ldots ||a_n = s + k \ mod \ m$, where $s + k \ mod \ m$ is the k-th state of the state space S starting from the state s, $s \in S$ is the first component of the key $(s, x_1|| \ldots ||x_n) \in K$ and $x_1|| \ldots ||x_n$ is the second one.

Finally, for every $w = a_1|| \ldots ||a_n, (a_1, \ldots, a_n) \in S$, put $\overline{w} = (a_1, \ldots, a_n)$. Then we can consider the double round $g' : K \times S \to U$ of $g : K \times S \to U$ (with $U = S$) such that for evey pair $k \in K, s \in S$, $g'(k, s) = g(k, \overline{g(k, s)})$. Similarly, for every $k > 2$, we can consider the k-times round $g'' : K \times S \to U$ of $g : K \times S \to U$ (with $U = S$) such that for evey pair $k \in K, s \in S$, $g''(k, s) = g(k, \overline{h(k, s)})$, where $h : K \times S \to U$ is a $(k-1)$-times round of $g : K \times S \to U$. This completes the proof. QED

By Proposition 1, Theorem 3, and Theorem 7, we can derive the following.

***Corollary* 8** Let $\mathcal{CBRNG} = (K, Z_J, S, f, U, g)$ be a counter based pseudorandom number generator with simple output multiplicity (i.e., $Z_J = \{0\}$) and assume that its output function is defined by the transition function of a given key automaton. Then $\mathcal{CBRNG}$ has a full cycle.

Proof. Of course, because the state transition f of $\mathcal{CBRNG}$ (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 $S = \{0, \ldots, p-1\}$, f has a full cycle. Moreover, by Theorem 4, the key automaton $\mathcal{B} = (\Sigma^n, (\Sigma^n)^{\log_2 n}, \delta_{\mathcal{B}}), n > 1$ is a permutation automaton, therefore, for every input sign $x \in (\Sigma^n)^{\log_2 n}$, $g_x : \Sigma^n \to \Sigma^n$ with $g_x(y) = \delta_{\mathcal{B}}(y, x)$ is a bijective mapping of Σ^n onto itself. By Proposition 1, that means that $\mathcal{CBRNG}$ (having a simple output multiplicity) has a full cycle. QED

Next we give an example and then we study the security of our $CBRNG$.

4 Example

In this section we show a simple example.

Consider the following transition table of an automaton $\mathcal{A} = (\{0, 1\}, \{0, 1\}^2, \delta)$:

δ	00	01	10	11
0	0	1	1	0
1	1	0	0	1

Let $n = 4$ and assume that all of $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4$ coincide with $\mathcal{A}$. Then $\log_2 n = \log_2 4 = 2$ and thus

$\mathcal{D}_1 = (\{1, \ldots, 4\}, \{(3, 1), (4, 2)\})$,

$\mathcal{D}_2 = (\{1, \ldots, 4\}, \{(2, 1), (4, 3)\})$.

Suppose that either a counter or a pseudorandom number generator sends an input $(1, 0, 1, 0, 1, 0, 1, 0)$ to the key automaton $\mathcal{B}$ which is the temporal product of $\mathcal{A}_{\mathcal{D}_1}$ and $\mathcal{A}_{\mathcal{D}_2}$. Assume that $\mathcal{B}$ is in the state $(0, 1, 1, 0)$.

Denote, in order, $\varphi_i, a_i, a'_i, x_i, i \in \{1, 2, 3, 4\}$ the feedback functions, the state components, the next state components, and the input components of $\mathcal{A}_{\mathcal{D}_1}$. Then $\varphi_1((0, 1, 1, 0), 1, 0, 1, 0) = (a_3 \oplus x_3, x_1) = (1 \oplus 1, 1) = (0, 1)$, $\varphi_2((0, 1, 1, 0), 1, 0, 1, 0) = (a_4 \oplus$

$x_4, x_2) = (0 \oplus 0, 0) = (0, 0)$, moreover $\delta(0, (0, 1)) = 1(= a'_1)$ and $\delta(1, (0, 0)) = 1(= a'_2)$, and thus $\varphi_3((0, 1, 1, 0), 1, 0, 1, 0) = (a'_1 \oplus x_1, x_3) = (1 \oplus 1, 1) = (0, 1)$, $\varphi_4((0, 1, 1, 0), 1, 0, 1, 0) = (a'_2 \oplus x_2, x_4) = (1 \oplus 0, 0) = (1, 0)$. Thus $\delta(1, (0, 1)) = 0(= a'_3)$ and $\delta(0, (1, 0)) = 1(= a'_4)$.

Next we denote by $\varphi_i, a_i, a'_i, x_i, i \in \{1, 2, 3, 4\}$ the feedback functions, the state components, the next state components, and the input components of $\mathcal{A}_{\mathcal{D}_2}$. Recall that (a_1, a_2, a_3, a_4) coincides with the new state of $\mathcal{A}_{\mathcal{D}_1}$. Then $(a_1, a_2, a_3, a_4) = (1, 1, 0, 1)$ and $(x_1, x_2, x_3, x_4) = (1, 0, 1, 0)$, where (x_1, x_2, x_3, x_4) consists of the last four components of the input $(1, 0, 1, 0, 1, 0, 1, 0)$ of the key automaton.

Then $\varphi_1((1, 1, 0, 1), 1, 0, 1, 0) = (a_2 \oplus x_2, x_1) = (1 \oplus 0, 1) = (1, 1)$, $\varphi_3((1, 1, 0, 1), 1, 0, 1, 0) = (a_4 \oplus x_4, x_3) = (1 \oplus 0, 1) = (1, 1)$, moreover $\delta(1, (1, 1)) = 1(= a'_1)$ and $\delta(0, (1, 1)) = 0(= a'_3))$, and thus $\varphi_2((1, 1, 0, 1), 1, 0, 1, 0) = (a'_1 \oplus x_1, x_2) = (1 \oplus 1, 0) = (0, 0)$, $\varphi_4((1, 1, 0, 1), 1, 0, 1, 0) = (a'_3 \oplus x_3, x_4) = (0 \oplus 1, 0) = (1, 0)$. Thus $\delta(1, (0, 0)) = 1(= a'_2)$ and $\delta(1, (1, 0)) = 0(= a'_4)$.

Hence the actual pseudorandom output is $(1, 1, 0, 0)$ which is also the next state.

5 Implementation

Next we give a detailed explanation of the enclosed pseudocode of our implementation. (See Algorithm ACBRNG.)

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 ($JSTATE$).

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 Kbyte.

The key automaton is a four-component temporal product of automata which are (D_1, D_2, D_3, D_4)-powers of the basic automaton. The digraphs D_1, D_2, D_3, D_4 are defined by the matrix $P[4][16]$. Each of the D_1, D_2, D_3, D_4 powers consists of sixteen copies of the basic automaton which has 256 states and 256 input signs. Thus, the transition matrix AUT of the basic automaton is a 256×256-type matrix, where each state and input sign can be represented by a 8-bit binary string.

The connection digraphs D_1, D_2, D_3, D_4 are stored in the four consecutive row vectors of the 4×16-type connection matrix P.

We will consider $ROUND = 1, 2, 3$ rounds of the output function of $CBRNG$.

The four row vectors of the 4×16-type input matrix $INPUT$ represent four consecutive input signs of the four (D_1, D_2, D_3, D_4)-powers, the key automaton of which the temporal product consists.

Thus the matrix $INPUT$ represents a single input sign of the key automaton.

The main structure of the implementation is the following.

1. Read the number SIZE of the input block, the input word INPUT of the key automaton, the transition matrix AUT of the basic automaton, and the initial (seed) state JSTATE of the key automaton.
2. Store the initial (seed) state JSTATE in a working storage ISTATE.
3. Generate $SIZE$ number of random blocks as follows.
4. Consider the $ISTATE$ as a 128-bit length binary number and fput
$ISTATE = ISTATE + 1 \; mod \; 2^{128}$.
5. Repeat the following procedure $ROUND$-times. We could not pass the NIST test for $ROUND = 1$ and $ROUND = 2$ but we were successful for $ROUND = 3$.
6. Each of the $ROUND$ number of repetitions operates on the actual value of the actual key automaton state ($ISTATE$) by the consecutive element (the consecutive input sign) of the input word ($INPUT$) .
7. The operation of the states of (D_1, D_2, D_3, D_4) – products by the consecutive input sign (i.e., the consecutive column vector of the matrix $INPUT$) determined by the transition table (AUT) of the basic automaton and the digraphs D_1, D_2, D_3, D_4 defined by the matrix $P[4][16]$.
8. Collect the records of the pseudorandom block OARRAY.
9. Output the consecutive pseudorandom block OARRAY.

6 Experimental results

We implemented Algorithm ACBRNG 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 NSIT SP-800-22 statistical randomness test.

6.1 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 [21]. In case of each statistical test a set of P-values was 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 is inconsistent with our null hypothesis, i.e. the 'hypothesis of randomness', so we reject it.

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

We used $\alpha = 0.01$ as it is common in such problems in cryptography and PRNG testing. An α of 0.01 indicates that one would expect 1 sequence in 100 sequences to be rejected under the null hypothesis. Hence a P-value exceeding 0.01 would mean that the sequence would be considered to be random, and a P-value less than or equal to 0.01 would lead to the conclusion that the sequence is non-random. One of the most important characteristics of a PRNG is the indistinguishability from true random sources. That is, the evaluation of their output utilizing statistical tests should not provide any means by which to distinguish them computationally from a truly random source.

6.2 Minimum rounds to achieve randomness

ACBRNG has a cycle length of 2^{128}, however this does not yet mean that ACBRNG is really producing good quality random numbers. Consider the simple generator

$$k \mod 2^{128}, \quad (k \in 0 \dots 2^{128}) \tag{1}$$

If we start $k = 0$ and increment by 1 then the generator has a 2^{128} cycle length, however it is not random at all. ACBRNG has a more complex structure, but statistical tests were needed to check for possible weaknesses. In order to test the quality of ACBRNG the NIST statistical tests were performed using the same parameters as for the AES candidates in order to achieve the most reliable and comparable results. First the input 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. Exact parameters can be seen in Table 1.

One Round ACBRNG We started the ACBRNG with a low entropy random seed. The running time was 4.5 sec using 8 parallel threads. Applying only one round ($ROUND = 1$ in line 25) ACBRNG did not pass the NIST requirements. More precisely, we have failed in almost every statistical test using one round. So one can conclude that only one round is not complex enough, and further investigation was needed. We would like to note, that surprisingly all Random Excursions tests from NIST were passed after one round.

Two Rounds ACBRNG Using $ROUND = 2$ surprisingly almost every statistical test was passed. The running time was 8.4 sec using 8 parallel threads. Only two non-overlapping templates were unsatisfied, which is quite a good achievement after two rounds. We did not expect such good quality random numbers and p-value distribution after two rounds. One can conclude that using only two rounds is enough to reach good quality random numbers and pass the NIST test.

Three Rounds ACBRNG After only three rounds ACBRNG did pass every requirement of the NIST statistical test suite. It has turned out that the output of the algorithm (using $ROUND = 3$) can not be distinguished in polynomial time from true random sources using the NIST statistical test. The running time was 12.25 sec using 8 parallel threads. The exact p-values of the evaluation of the ACBRNG 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 provide no additional information about the PRNG. We have also shown that the proportions of binary sequences which passed the 0.01 level lie in the required confidence interval (see e.g. [21]).

7 Conclusion and further research

In this paper a full cycle length pseudorandom number generator (ACBRNG) based on compositions of automata was presented. We have seen that it produces promising results in terms of statistical randomness and passed all statistical tests included in the NIST test suite. We can see that the running time of the generator is efficient enough for practical use. In order to consider this generator cryptographically secure, formal verification of its security would be necessary which is a direction that might be worth investigating.

References

[1] Bao, F.: Cryptoanalysis of a partially known cellular automata cryptosystem. IEEE Trans. on Computers, 53 (2004), 1493-1497; https://doi.org/10.1109/TC.2004.94

[2] Bilan, S., Bilan, M., Bilan, S: Novel pseudorandom sequence of numbers generator based cellular automata. Information Technology and Security, Vol. 3(1), 2015, pp. 38-50. https://doi.org/10.20535/2411-1031.2015.3.1.57710

[3] Bhattacharjee, K., Das, S., Paul, D.: Pseudo-random Number Generation using a 3-state Cellu lar Automaton. Internat. J. of Modern Physics, vol 28, No 6,

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	Prop	Test name
24	34	30	31	25	30	26	33	30	37	0.828458	297/300	Frequency
34	29	32	27	21	26	20	35	31	45	0.068287	296/300	BlockFrequency
29	32	36	31	31	24	31	27	29	30	0.964295	298/300	CumulativeSums
26	30	31	30	33	27	24	38	28	33	0.840081	295/300	CumulativeSums
32	22	25	35	36	25	32	30	41	22	0.198690	300/300	Runs
34	34	34	31	37	23	33	23	29	22	0.437274	298/300	LongestRun
22	26	23	35	35	32	34	39	29	25	0.334538	296/300	Rank
33	28	31	29	33	30	34	31	25	26	0.973936	296/300	FFT
30	30	21	35	40	29	29	28	25	33	0.514124	296/300	NonOverLappingTemp
43	30	24	29	34	27	32	22	31	28	0.339799	296/300	NonOverLappingTemp
.	.	.	.	.	.	.	.	.	.	.	.	NonOverLappingTemp
...	...	...	...	...	...	...	...	...	...	...	...	NonOverLappingTemp
22	21	30	44	38	32	23	31	35	24	0.043745	295/300	OverlappingTemp
32	44	35	29	28	26	20	23	34	29	0.132132	298/300	Universal
28	27	18	36	40	26	33	37	23	32	0.122325	297/300	ApproximateEntropy
17	15	23	20	20	21	16	27	17	18	0.707944	194/194	RandomExcursions
22	20	21	13	25	16	19	17	21	20	0.791218	193/194	RandomExcursions
.	.	.	.	.	.	.	.	.	.	.	.	RandomExcursions
...	...	...	...	...	...	...	...	...	...	...	...	RandomExcursions
22	17	15	16	23	23	20	21	23	14	0.729339	192/194	RandomExcursionsV
18	18	17	19	23	27	19	17	17	19	0.838872	193/194	RandomExcursionsV
.	.	.	.	.	.	.	.	.	.	.	.	RandomExcursionsV
...	...	...	...	...	...	...	...	...	...	...	...	RandomExcursionsV
31	28	27	22	27	39	26	36	35	29	0.514124	296/300	Serial
30	32	22	36	32	22	33	30	35	28	0.637119	294/300	Serial
32	28	31	26	34	24	32	37	36	20	0.449672	296/300	LinearComplexity

Algorithm ACBRNG

```
1: procedure ACBRNG(SIZE, INPUT, AUT, JSTATE)          ▷ 1. Read the input parameters.
2:     for i = 0 → 15 do                               ▷ 2. Put the seed into working storage.
3:         ISTATE[i] ← JSTATE[i]                       ▷ 3. JSTATE is the initial (seed) state of the key automaton.
4:     end for
5:     for kk = 0 → SIZE do                            ▷ 4. SIZE number of pseudorandom blocks are generated
6:         for m = 0 → 127 do
7:             x ← 0
8:             for j = 15 → 0 do
9:                 if x = 0 then
10:                    if ISTATE[j] = 255 then
11:                        ISTATE[j] ← 0
12:                    else
13:                        ISTATE[j] ← ISTATE[j] + 1
14:                        x ← 1
15:                    end if
16:                end if
17:                STATE[j] ← ISTATE[j]
18:            end for
19:            for f = 0 → ROUND do                    ▷ 5. Passes NIST test with ROUND = 3.
20:                for i = 0 → 3 do                    ▷ 6. Key automaton state transition.
21:                    for j = 0 → 15 by 2 do          ▷ 7. D1, D2, D3, D4-power state transitions.
22:                        k ← P[i][j]
23:                        l ← P[i][j + 1]
24:                        a1 ← STATE[k]
25:                        a2 ← STATE[l] ⊕ INPUT[l][i]
26:                        STATE[k] ← AUT[a1][a2]
27:                        a1 ← STATE[l]
28:                        a2 ← STATE[k] ⊕ INPUT[k][i]
29:                        STATE[l] ← AUT[a1][a2]
30:                    end for
31:                end for
32:            end for
33:            for i = 0 → 15 do                       ▷ 8. Collect the records of the pseudorandom block OARRAY
34:                OARRAY[m][i] ← STATE[i]
35:            end for
36:        end for
37:        PRINT(&OARRAY)                              ▷ 9. Print the next random block.
38:    end for
39: end procedure
```

ppp. 1-23, 2017, https://doi.org/10.1142/S0129183117500784

[4] Chakraborty,K. and . Chowdhury, D. R. : CSHR: Selection of cryptographically suitable hybrid cellular automata rule. International Conference on Cellular Automata for Research and Industry, ACRI, Springer, pp. 591-600, 2012. https://doi.org/10.1007/978-3-642-33350-7_61

[5] Dogaru, R. and Dogaru,I.: FPGA implementation and evaluation of two cryptographically secure hybrid cellular automata. Proc. Communications (COMM) 2014, 10th International Conference on Communications, pp. 1-4, 2014. https://doi.org/10.1109/ICComm.2014.6866740

[6] Dömösi, P., Gáll, J., Horváth, G., Tihanyi, N. Some Remarks and Tests on the Dh1 Cryptosystem Based on Automata Compositions Informatica (Slovenia), vol 43, 2 (2019), pp. 199-207. https://doi.org/10.31449/inf.v43i2.2687

[7] Dömösi, P. and Nehaniv, C. L. Algebraic theory of automata networks. An introduction. SIAM Monographs on Discrete Mathematics and Applications, 11. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2005. https://doi.org/10.1137/1.9780898718492

[8] Guan, S. U. and Tan, S. K.: Pseudorandom number generation with self-programmable cellular automata. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 23, pp. 1095-1101, 2004. https://doi.org/10.1109/TCAD.2004.829808

[9] Guan, S.-U. and Zhang, S.: Pseudorandom number generation based on controllable cellular automata. Future Generation Computer Systems, vol. 20, pp. 627-641, 2004. https://doi.org/10.1016/S0167-739X(03)00128-6

[10] Guan, P.: Cellular automaton public key cryptosystem. Complex Systems, 1 (1987), 51-56].

[11] Hoe, D. H. K., Comer, J. M., Cerda, J. C., Martinez, C. D., Shirvaikar,M. V.: Cellular Automata-Based Parallel Random Number Generators Using FPGAs. International Journal of Reconfigurable Computing, Vol. 2012, 2012, pp. 1-13, Article ID 219028 https://doi.org/10.1155/2012/219028

[12] Hortensius, P. D., Mcleod, R. D., Pries, W.,Miller, D M., and Card, H C.: Cellular Automata- Based Pseudorandom Number Generators for Built-In self-Test, IEEE transactions on Computer-Aided Design, vol. 8, pp 842-859,1989. https://doi.org/10.1109/43.31545

[13] Kang, B., Lee, D., and Hong, C.: High-Performance Pseudorandom Number Generator Using Two-Dimensional Cellular Automata. 4th IEEE International Symposium on Electronic Design, Test and Applications (delta 2008), Hong Kong, 2008, pp. 597-602. https://doi.org/10.1109/DELTA.2008.46

[14] Kar, M., Rao, D. C., Rath, A. K.: Generating PNS for Secret Key Cryptography Using Cellular Automaton. International Journal of Advanced Computer Science and Applications, Vol. 2, No. 5, 2011, pp. 101-105. https://doi.org/10.14569/IJACSA.2011.020517

[15] Madghuri, A.: Hybrid Cellular Automata-Based Pseudo Random Sequence Generator for BIST Implementation. International Journal of Research Studies in Science, Engineering and Technology Volume 2, Issue 9, September 2015, pp. 72-76 ISSN 2349-4751 (Print) & ISSN 2349-476X (Online)

[16] Martin, B., Sole, P.: Pseudo-random Sequences Generated by Cellular Automata. International Conference on Ralations, Orders and Graphs: Interaction with Computer Scince, May 2008, Mandia, Tunisia, Nouha editions, 2008, pp. 401-410.

[17] Meier, W. and Staffelbach, O.: Analysis of pseudo random sequences generated by cellular automata. In: Davies, D. W. (ed.), Proc. Conf. Advances in Cryptology – EUROCRYPT '91, Workshop on the Theory and Application of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, LNCS 547 Springer-Verlag, Berlin, 1991, 186-199 https://doi.org/10.1007/3-540-46416-6

[18] Menezes, A., van Oorschot,P., and Vanstone, S.: Handbook of Applied Cryptography, CRC Press, 1996.

[19] Ping,P., Xu, F., and Wang, X.-J.: Generating high-quality random numbers by next nearest-neighbor cellular automata. Advanced material Research, vol. 765, pp. 1200-1204, 2013. https://doi.org/10.4028/www.scientific.net/AMR.765-767.1200

[20]] Ruboi, C. F., Encinas, L. H., White, S. H., del Rey, A. M., Sancher, R.: The use of Linear Hybrid Cellular Automata as Pseudorandom bit Generators in Cryptography. Neural Parallel & Scientific Comp. 12(2), 2004, pp. 175-192. http://hdl.handle.net/10261/21253

[21] Rukhin, A., Soto, J., Nechvatal, J.,Smid, M., Barker, E., Leigh, S., Levenson, M.,Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S.: NIST Special Publication 800-22 (2010).: A Statistical Test Suite for Random and Pseudo Random

Number Generators for Cryptographic Applications. *National Institute of Standards and Technology*, https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf, downloaded in March 2020. https://doi.org/10.6028/NIST.SP.800-22r1a

[22] Salmon, J., Moares, M., Dror, R., Shaw, D. Parallel random numbers: as easy as $1, 2, 3$. Proc. 2011 Intern. Conf. for High Performance Computing, Networking, Storage and Analysis, Article No. 16. doi:10.1145/2063384.2063405. https://doi.org/10.1145/2063384.2063405

[23] Seredynski, F., Bouvry, P., and Zomaya A. Y.: Cellular automata computations and secret key cryptography. Parallel Computing, vol. 30, pp. 753-766, 2004. https://doi.org/10.1016/j.parco.2003.12.014

[24] Shin,SH., Kim,DS., Yoo, KY. : A 2-Dimensional Cellular Automata Pseudorandom Number Generator with Non-linear Neighborhood Relationship. In: Benlamri R. (eds) Networked Digital Technologies. NDT 2012. Communications in Computer and Information Science, vol 293. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-30507-8_31

[25] Sipper, M., Tomassini, M. Generating parallel random number generators by cellular programming. International Journal of Modern Physics C. 7 (2), pp. 181–190, 1996. https://doi.org/10.1142/S012918319600017X

[26] Sukhinin, B.M.: High-speed pseudorandom sequence generators based on cellular automata. Applied discrete mathematics, No 2, 2010, pp. 34 – 41. https://doi.org/10.17223/20710410/8/5

[27] Sukhinin B.M.: Development of generators of pseudorandom binary sequences based on cellular automata. Science and education, No 9, 2010, pp. 1 – 21.

[28] Tomassini, M., Sipper, M., and Perrenoud, M.: On the Generation of High-Quality Random Numbers by Two-Dimensional Cellular Automata, IEEE Transactions on Computers, vol. 49, pp. 1146-1151, 2000. https://doi.org/10.1109/12.888056

[29] Wolfram, S.,: Cryptography with Cellular Automata. In: C. W. Hugh, ed., Proc. Conf. Advances in Cryptology—CRYPTO'85, Santa Barbara, Calif., USA, Aug. 18-22, 1985, LNCS 218, Springer-Verlag, Berlin, 1986, pp. 429-432. https://doi.org/10.1007/3-540-39799-X_32

[30] Wolfram, S.: Random Sequence Generation by Cellular Automata. Advances in Appl. Math., vol. 7, 1986, pp. 429-432. https://doi.org/10.1016/0196-8858(86)90028-X