Some Remarks and Tests on the DH1 Cryptosystem Based on Automata Compositions

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

József Gáll, 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

Norbert Tihanyi
Faculty of Informatics, Eötvös Loránd University
H-1117 Budapest, Pázmány Péter setány I/C, Hungary
E-mail: tihanyi.pgp@gmail.com

Keywords: automata network, NIST test, block cipher, statistics

Received: February 17, 2019

In this paper we discuss NIST test results of a previously introduced cryptosystem based on automata compositions. We conclude that the requirements of NIST test are all fulfilled by the cryptosystem.

Povzetek: Analiziran je kriptirni sistem DH1 na osnovi končnih avtomatov s testom NIST.

1 Introduction and problem statement

Dömösi and Horváth in their previous works (see [Dömösi and Horváth, 2015a] and [Dömösi and Horváth, 2015b]) introduced new block ciphers based on Gluškov-type product of automata. In what follows we will refer to the cipher in [Dömösi and Horváth, 2015a] as the first Dömösi-Horváth cryptosystem, or in short, DH1-cipher, whereas to the cipher in [Dömösi and Horváth, 2015b] as the second Dömösi-Horváth cryptosystem, or in short, DH2-cipher. In this paper we investigate some properties of the DH1-cipher. However, we do not discuss all details of definition and motivation regarding DH1-chipers in this paper.

Both systems use the following simple idea: consider a giant-size permutation automaton such that the set of states and the set of inputs consisting of all given length of strings over a non-trivial alphabet as all possible plaintext/ciphertext blocks. Moreover consider a cryptographically secure pseudo random number generator with large periodicity having the property that, getting its really random kernel, it serves a sequence of pseudo random strings as inputs for the automaton. For each plaintext block the system calculates the new state into which the actual pseudorandom string takes the automaton from the state which is identified as the actual plaintext block. The string – identified as the new state– will be the ciphertext block ordered to the considered plaintext block. Of course, the ciphertext will be the concatenation of the generated ciphertext blocks. The giant size of the automaton makes it infeasible to break the system by brute-force method.

For all notions and notations not defined in this paper we refer to the monographs [Dömösi and Nehaniv, 2005, Mezenes and Vanstone, 1996]. The cryptosystem discussed here is a block cipher. Since the key automaton is a permutation automaton, for every ciphertext there exists exactly one plaintext making the encryption and decryption unambiguous. Moreover, there is a huge number of corresponding encoded messages to each plaintext so that several encryptions of the same plaintext yield several distinct ciphertexts.

Given the cryptosystem DH1-cipher described above a natural question is the investigation of the statistical properties of the system from many perspectives. For instance, the avalanche effect of the system –as a natural property required in the profession– may be tested by several classical hypothesis tests. Some early results are given in [Dömösi et al., 2017] where they confirm that the avalanche effect is fulfilled. However, further tests can and should also be used, in particular the ones used for testing whether the output of it can be distinguished from 'true' random sources. That is why we turned to the well known
NIST package of statistical tests in this paper, which can be considered as a 'standard' in the profession for such purposes. Our main aim is to give the results of the NIST test regarding the cryptosystem at issue (Section 5). For this we describe the system (Section 3) together with some theoretical background (Section 2), as well as the necessary details, of course, of our experimental analysis done for the tests (Section 4). We show in this paper that the system we discuss has passed all statistical tests in the NIST package.

2 Theoretical background

The automata are systems that can be used for the transmission of information of certain type. In wider sense, every system that accepts signals from its environment and, as a result, changes its internal state, can be considered as an automaton. By an automaton we mean a deterministic finite automaton without outputs. The automaton $A = (A, \Sigma, \delta)$ consists of the finite set of states $A$, the finite set of input signals $\Sigma$, and the transition function $\delta$, which is often written in a matrix form. The transition matrix of the automaton $A = (A, \Sigma, \delta)$ consists of its states such that it has as many rows as input signals, and there are as many columns as states of the automaton. For the sake of simplicity we assume that $A$ and $\Sigma$ are ordered sets. The $j$-th element of the $i$-th row of the transition matrix will be the state which is assigned by the transition function to the pair consisting of $j$-th state and $i$-th input signal. We say about this element $a$ of the $i$-th row and $j$-th column of the transition matrix that the $i$-th input signal takes the automaton from its $j$-th state to state $a$. (In fact, in this case it is also usual to say that the automaton goes from its $j$-th state to state $a$ by the effect of the $i$-th input signal.) The rows of the transition matrix can be identified with the input signals of the automaton, and its columns with its states, while the transition matrix itself with the transition.

If all the rows of the transition matrix are permutations of the state set then we have a permutation automaton.

**Lemma 1.** An automaton $A = (A, \Sigma, \delta)$ is a permutation automaton if and only if for any $a, b \in A, x \in \Sigma, \delta(a, x) = \delta(b, x)$ implies $a = b$.

**Proof.** Suppose that $A$ is a permutation automaton. Then all rows in its transition matrix are permutations of the state set. But then none of the rows of the transition matrix has a repetition. Therefore, for any states $a, b \in A$ and input $x \in \Sigma$, $\delta(a, x) = \delta(b, x)$ implies $a = b$. Conversely, assume that for any states $a, b \in A$ and input $x \in \Sigma$, $\delta(a, x) = \delta(b, x)$ implies $a = b$. Then none of the rows of the transition matrix has a repetition. Therefore all of its rows are permutations of the state set. This completes the proof.

The Gluškov-type product of the automaton $A_i$ with respect to the feedback functions $\varphi_i$ ($i \in \{1, \ldots, n\}$) is defined to be the automaton $A = A_1 \times \cdots \times A_n(\varphi_1, \ldots, \varphi_n)$ with state set $A = A_1 \times \cdots \times A_n$, input set $\Sigma$, transition function $\delta$ 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 $x \in \Sigma$ (see also Figure 1). In particular, if $A_1 = \ldots = A_n$ then we say that $A$ is a Gluškov-type power.

We shall use the feedback functions $\varphi_i, i = 1, \ldots, n$ in an extended sense as mappings $\varphi_i^*: A_1 \times \cdots \times A_n \times \Sigma^*$, where $\varphi_i^*(a_1, \ldots, a_n, \lambda) = \lambda$ and $\varphi_i^*(a_1, \ldots, a_n, px) = \varphi_i^*(a_1, \ldots, a_n, p)\varphi_i(\delta_1(a_1, \varphi_1(a_1, \ldots, a_n, p)), \ldots, \delta_n(a_n, \varphi_n(a_1, \ldots, a_n, p)), x), a_i \in A_i, \ i = 1, \ldots, n, p \in \Sigma^*, \ x \in \Sigma$. In the sequel, $\varphi_i^*, i \in \{1, \ldots, n\}$ will also be denoted by $\varphi_i$.

Next we define the concept of temporal product of automata. It is a model for multichannel automata networks where the network may cyclically change its internal structure during its work on each channel.

Let $A_t = (A, \Sigma_t, \delta_t), t = 1, 2$ be automata having a common state set $A$. Take a finite nonvoid set $\Sigma$ and a mapping $\varphi$ of $\Sigma$ into $\Sigma_1 \times \Sigma_2$. Then the automaton $A = (A, \Sigma, \delta)$ is a temporal product ($t$-product) of $A_1$ by $A_2$ with respect to $\Sigma$ and $\varphi$ if for any $a \in A$ and $x \in \Sigma$, $\delta(a, x) = \delta_2(\delta_1(a, x_1), x_2)$, where $(x_1, x_2) = \varphi(x)$ (see also Figure 2). The concept of temporal product is generalized in the natural way to an arbitrary finite family of $n > 0$ automata $A_t, t = 1, \ldots, n$, all with the same state set $A$, for any mapping $\varphi: \Sigma \to \prod_{i=1}^n \Sigma_i$, by defining $\delta(a, x) = \delta_n(\cdots \delta_2(\delta_1(a, x_1), x_2), \cdots, x_n)$ when $\varphi(x) = (x_1, \ldots, x_n)$. In particular, a temporal product of automata with a single factor is just a (one-to-many) relabeling of the input letters of some input-subautomaton of its factor.

**Lemma 2.** Every temporal product of permutation automata is a permutation automaton.

**Proof.** It is clear from the above mentioned remark that every temporal product of permutation automata with a single factor is a permutation automaton. Now let $A_i = (A, \Sigma_i, \delta_i), t = 1, 2$ be permutation automata with the same state set $A$. Consider a temporal product of $A_1$ and $A_2$ with respect to an arbitrary input set $\Sigma$ and mapping $\varphi: \Sigma \to \prod_{i=1}^n \Sigma_i$. Prove that for any $a, b \in A, z \in \Sigma$ with $\varphi(z) = (x, y), \delta_2(\delta_1(a, x), y) = \delta_2(\delta_1(b, y), x)$ implies $a = b$.
Indeed, let $\delta_1(a,x) = e$ and $\delta_1(b,x) = d$. Recall that $A_2$ is a permutation automaton. Therefore, by Lemma 1, $\delta_2(c,y) = \delta_2(d,y)$ implies $c = d$. On the other hand, $A_1$ is also a permutation automaton. Thus, by Lemma 1, $c = d$ with $\delta_1(a,x) = e$ and $\delta_1(b,x) = d$ imply $a = b$. Applying Lemma 1 again, we receive that the temporal product of $A_1$ and $A_2$ with respect to $\Sigma$ and $\varphi$ is a permutation automaton. Therefore our statement holds for all temporal products having two factors. Now we consider a temporal product of permutation automata $A_1, \ldots, A_n, n > 2$ with respect to a given set $\Sigma$ and mapping $\varphi$.

Define the mappings $\varphi_1 : \Sigma \rightarrow \Sigma_1 \times \Sigma_2, \varphi_2 : \Sigma \rightarrow (\Sigma_1 \times \Sigma_2) \times \Sigma_3, \ldots, \varphi_{n-1} : \Sigma \rightarrow \ldots (\Sigma_1 \times \Sigma_2) \times \ldots \times \Sigma_n$ with $\varphi_1(x) = (x_1, x_2), \varphi_2(x) = ((x_1, x_2), x_3), \ldots, \varphi_{n-1}(x) = ((\ldots((x_1, x_2), x_3), \ldots), x_n)$ whenever $\varphi(x) = (x_1, x_2, n)$. Let $B_1$ denote the temporal product of $A_1$ and $A_2$ with respect to $\Sigma$ and $\varphi_1$, and $B_2$ denote the temporal product of $B_1$ and $A_3$ with respect to $\Sigma$ and $\varphi_2$. Then using the fact that our statement holds for all temporal products with two factors we obtain that all of $B_1, \ldots, B_{n-1}$ are permutation automata. On the other hand, it is clear that $B_{n-1}$ is equal to the temporal product of permutation automata $A_1, \ldots, A_n$ with respect to $\Sigma$ and $\varphi$. Thus the proof is complete.

A function $f : X_1 \times \cdots \times X_n \rightarrow Y$, we say that $f$ is really independent of its i-th variable if for every pair $(x_1, \ldots, x_n), (x_1', \ldots, x_i', x_{i+1}, \ldots, x_n) \in X_1 \times \cdots \times X_n$ $f(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots, x_n)$. Otherwise we say that $f$ really depends on its i-th variable.

A (finite) directed graph (or, in short, a digraph) $D = (V, E)$ (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. An edge $(v, v') \in E$ is said to have a source $v$ and a target $v'$. Moreover, we say that $v \in V$ is a source if there exists a $v' \in V$ having $(v, v') \in E$, and $v' \in V$ is a target if there exists a $v \in V$ with $(v, v') \in E$. The pair $(v, v'), (v'', v''')$ is called a branch if $v \neq v''$ and $v' \neq v'''$. In addition, the pair $(v, v'), (v'', v''')$ is called a collapse if $v \neq v''$ and $v' = v'''$. If $|V| = n$ then we also say that $D$ is a digraph of order $n$. If $V$ can be decomposed into two disjoint (nonempty) subsets $V_1, V_2$ such that $V_1$ is the set of all targets and $V_2$ is the set of all sources then we say that $D$ is a bipartite digraph.

If the bipartite graph $D$ has neither branches nor collapses then we say that $D$ is a simple bipartite digraph.

Let $\Sigma$ be the set of all binary strings with a given length $\ell > 0$ and let $n$ be a positive integer power of 2, let $A_1 = (\Sigma, \Sigma \times \Sigma, \delta_{A_1})$ be a permutation automaton such that for every $a, x, x', y, y' \in \Sigma, \delta_{A_1}(a, (x, y)) \neq \delta_{A_1}(a, (x', y)), \delta_{A_1}(a, (x, y)) \neq \delta_{A_1}(a, (x', y'))$, and let $A_i = (\Sigma, (\Sigma \times \Sigma, \delta_{A_i}), i) = 2, \ldots, n$ be state-isomorphic copies of $A_1$ such that $A_1, \ldots, A_n$ are not necessarily pairwise distinct, and let $n$ be a power of 2. Consider the following simple bipartite digraphs:

$D_1 = \{(1, \ldots, n), \{(n/2 + 1, 1), (n/2 + 2, 2), \ldots, (n, n/2)\}\},$

$D_2 = \{(1, \ldots, n), \{(n/4 + 1, n/2 + 1), (n/4 + 2, 2n + 2)\}, \{(n, 3n/4)\}\},$

$\bar{D}_{\log_2 n - 1} = \{(1, \ldots, n), \{(3, 1), (4, 2), (7, 5), \ldots, (8, 6), (n - 1, n - 3), (n, n - 2)\}\},$

$\bar{D}_{\log_2 n} = \{(1, \ldots, n), \{(2, 1), (4, 3), \ldots, (n, n - 1)\}\},$

$\bar{D}_{\log_2 n + 1} = D_1,$

$\bar{D}_{\log_2 n + 2} = \bar{D}_{\log_2 n}.$

For every digraph $D = (V, E)$ with $D \in \{D_1, \ldots, D_{2\log_2 n}\}$ let us define the Glushkov-type product, called two-phase $D$-product, $A_D = A_1 \times \cdots \times A_n(\Sigma^n, \varphi_1, \ldots, \varphi_n)$ of $A_1, \ldots, A_n$ so that for every $(a_1, \ldots, a_n), (x_1, \ldots, x_n) \in \Sigma^n, i \in \{1, \ldots, n\}, \varphi_i(a_1, \ldots, a_n, (x_1, \ldots, x_n)) = (a_i \oplus x_i, x_i)$, if $(i, i) \in E$, and $a_i \oplus x_i$ is the bitwise addition modulo 2 of $a_i$ and $x_i$.

$\bar{A}_{\log_2 n} = \bar{D}_{\log_2 n}.$

1Let $B = (\Sigma^n, \varphi) : (\Sigma^n)^{2\log_2 n} \rightarrow (\Sigma^n)^{2\log_2 n}$ be the temporal product of $A_D$ with respect to $(\Sigma^n)^{2\log_2 n}$ and the identity mapping $\varphi : (\Sigma^n)^{2\log_2 n} \rightarrow (\Sigma^n)^{2\log_2 n}$.

1 We remark that for every $j \in V_1$ there exists exactly one $i \in V_1$ with $(j, i) \in E$, and conversely, for every $i \in V_1$ there exists exactly one $j \in V_2$ with $(j, i) \in E$. Therefore, all of $\varphi_1, \ldots, \varphi_n$ are well-defined.

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

\textbf{Theorem 1}. Every key-automaton is a permutation automaton.

\textbf{Proof}. Let $B = (\Sigma^n, \varphi) : (\Sigma^n)^{2\log_2 n} \rightarrow (\Sigma^n)^{2\log_2 n}$ be a key-automaton. By definition, it is a temporal product of automata $A_{D_1}, \ldots, A_{D_{2\log_2 n}}$ with respect to $(\Sigma^n)^{2\log_2 n}$ and
we obtain that
\[
\delta_{D} = (\Sigma \oplus_{n}^{2}) \rightarrow (\Sigma \oplus_{n}^{2})
\]
as defined above. By Lemma 2, it is enough to prove that each of
\[
A_{D1}, \ldots, A_{D_{2^{log_{2}n}}}
\]
is a permutation automaton.

Consider an automaton \(A_{D} = (\Sigma^{n}, \Sigma^{n}, \delta_{D})\) with \(A_{D} \in \{A_{D1}, \ldots, A_{D_{2^{log_{2}n}}}\}\) and the simple bipartite digraph \(D = (V, E)\) assigned to \(A_{D}\). Let \(V_{1}\) denote the set of targets and \(V_{2}\) denote the set of sources of \(D\) as before.

By Lemma 1 it is enough to prove that for any states \((a_{1}, \ldots, a_{n})\), \((a'_{1}, \ldots, a'_{n})\) \(\in \Sigma^{n}\) and input \((x_{1}, \ldots, x_{n})\) \(\in \Sigma^{n}\),
\[
\delta_{D}((a_{1}, \ldots, a_{n}), (x_{1}, \ldots, x_{n})) = \delta_{D}((a'_{1}, \ldots, a'_{n}), (x_{1}, \ldots, x_{n})) \implies (a_{1}, \ldots, a_{n}) = (a'_{1}, \ldots, a'_{n}).
\]

Suppose \(\delta_{D}((a_{1}, \ldots, a_{n}), (x_{1}, \ldots, x_{n})) = \delta_{D}((a'_{1}, \ldots, a'_{n}), (x_{1}, \ldots, x_{n})) = (b_{1}, \ldots, b_{n})\) for some state \((b_{1}, \ldots, b_{n})\) of \(A_{D}\) and let \((i, j) \in E\). Observe that for every \(i \in V_{1}\) there exists exactly one \(j \in V_{2}\) with \((j, i) \in E\), and vice versa, for every \(j \in V_{2}\) there exists exactly one \(i \in V_{1}\) with \((j, i) \in E\). This means that the transitions in the \(i\)-th and \(j\)-th component automata depend only on the \(i\)-th and \(j\)-th state and input components.

Then, by the effect of its input \((a_{i} \oplus x_{i}, x_{i})\) the \(i\)-th component of \(A_{D}\) goes from its state \(a_{i}\) to state \(b_{i}\), and similarly, by the effect of its input \((b_{i} \oplus x_{i}, x_{i})\) the \(j\)-th component of \(A_{D}\) goes from its state \(a_{j}\) to state \(b_{j}\).

But then by the effect of its input \((a'_{i} \oplus x_{i}, x_{i})\), the \(i\)-th component of \(A_{D}\) goes from its state \(a'_{i}\) to state \(b_{i}\), and similarly, by the effect of its input \((b_{i} \oplus x_{i}, x_{i})\), the \(j\)-th component of \(A_{D}\) goes from its state \(a'_{j}\) to state \(b_{j}\).

Recall that \(A_{j}\) is a permutation automaton. Therefore, applying Lemma 1, \(a_{j} = a'_{j}\). Therefore, using our previous assumptions we can derive that by the effect of its input \((a_{j} \oplus x_{j}, x_{j})\) the \(i\)-th component of \(A_{D}\) goes from its state \(a_{i}\) to state \(b_{i}\). On the other hand, we assumed that by the effect of its input \((a_{j} \oplus x_{j}, x_{j})\), the \(i\)-th component of \(A_{D}\) goes from its state \(a_{i}\) to state \(b_{i}\). Applying Lemma 1 again we obtain that \(a_{i} = a'_{i}\).

Applying the above treatment to every \((i, j) \in E\), we receive \((a_{1}, \ldots, a_{n}) = (a'_{1}, \ldots, a'_{n})\). This completes the proof.

The basic idea of DH1 cryptosystem is to use a finite automaton and a pseudo random generator. The set of states of the automaton consists of all possible plaintext/ciphertext blocks and the input set of the automaton contains all possible pseudo random blocks. The size of the pseudo random blocks are the same as the size of the plaintext/ciphertext blocks. For each plaintext block the pseudo random generator generates the next pseudo random block and the automaton transforms the plaintext block into a ciphertext block by the effect of the pseudo random block.

The key is the transformation matrix of the automaton.

It is easy to see that the key must be a permutation automaton, since this property grants an unambiguous decryption. This condition is satisfied by Theorem 1.

On the other hand we can have more than one corresponding ciphertext for each plaintext even if we use the same key-automaton. The reason for this is that we can change the pseudo random numbers generated by the pseudo random generator. We can save a secret number \(n\) as a part of the key and before encryption we can choose a (public) random number \(m\). This number \(m\) will be the first block of the ciphertext, and before encryption and decryption, the seed of the pseudo random number generator can be calculated with an XOR operation from \(n\) and \(m\) \((n \oplus m)\). This way each encryption process uses different pseudo random numbers and results different ciphertext for the same plaintext.

The problem with this idea is the following. Modern block ciphers operate on fixed-length groups of bits called blocks. The size of the blocks is at least 128 bits (16 bytes), so the size of the transition matrix of the automaton is huge, namely \(2^{128} \times 2^{128} \times 16\) bytes, which is impossible to be stored in the memory or on a hard disk. The solution is to use an automata network. Gluškov-type product of automata consists of smaller component automata and it is able to simulate the operation of a huge automaton. In this case we should store only the transition matrix of the isomorphic component-automata, the structure of the composition and the secret number \(n\) to calculate the seed of the pseudo random number generator.

### 3 Encryption and decryption

A symmetric cryptosystem consists of the following:

- a set of plaintexts \(\mathcal{P}\),
- a set of ciphertexts \(\mathcal{C}\),
- a key space \(\mathcal{K}\),
- an encryption function \(e : \mathcal{P} \times \mathcal{K} \rightarrow \mathcal{C}\), and
- a decryption function \(d : \mathcal{C} \times \mathcal{K} \rightarrow \mathcal{P}\).

Furthermore, the following property must hold for each \(x \in \mathcal{P}\) and \(k \in \mathcal{K}\):

\[
d(e(x, k), k) = x.
\]

Moreover, the cryptosystem is called approved block cipher if and only if the elements of the set of plaintexts and the set of ciphertexts are at least 128 bit long \((|\mathcal{P}| \geq 2^{128} \text{ and } |\mathcal{C}| \geq 2^{128})\).

Our cryptosystem is a block cipher one. Both of the encryption and decryption apparatus have a pseudo random generator and a key-automaton.

The encryption procedure is the following. Before the encryption procedure, the pseudo random generator gets its initialization vector as a true random string \(r_{1} \ldots r_{n} \in \Sigma^{n}\), where the pseudo random alphabet \(\Sigma\) is also the plaintext and the ciphertext alphabet simultaneously. This initialization vector will also be the first block of the ciphertext.

Then the apparatus reads the plaintext block-by-block and, after reading the next plaintext block \(a_{1} \ldots a_{n} \in \Sigma^{n}\) (the first block first), it generates the second, third, and the further blocks of the ciphertext in the following way.

The apparatus takes the key-automaton \(B = (\Sigma^{n}, (\Sigma^{n})^{2log_{2}n}, \delta_{B})\) into the state \(a_{1} \ldots a_{n} \in \Sigma^{n}\).
which coincides with the actual one, i.e. the last received plaintext block.

Next the pseudo random number generator generates a $2\log_2 n$ long number of pseudo random sequences $w_1, \ldots, w_{2\log_2 n} \in \Sigma^n$ such that each of them takes the next temporal component (the first one first) $A_D = (\Sigma^n, \Sigma^n, \delta_D)$ ($A_D \in \{ A_{D_1}, \ldots, A_{D_{2\log_2 n}} \}$) of the key automaton into the state $a_k \cdots a_k n = \delta_D(a_k-1 \cdots a_k-1, n, w_k), k = 1, \ldots, 2\log_2 n$, where $a_{0,1} \cdots a_{0,n}$ denotes the actual plaintext block.

The last state $a_{0,1} \cdots a_{0,n}$ will be the generated ciphertext block of the plaintext block $a_1 \cdots a_n$.

The $i$-th transition $a_{1,1} \cdots a_{i-1,1} n, w_i$ will be performed in the following way.

Recall again that $D$ is a Čukić product $A_D = A_1 \times \cdots \times A_n(\Sigma^n, (\varphi_1, \ldots, \varphi_n))$ of appropriate permutation automata $A_m = (\Sigma, \Sigma^2, \delta_m), m = 1, \ldots, n$ that are state isomorphic to each other so that for an appropriate bipartite digraph $D = (V, E)$ with the set $V_1$ of targets and $V_2$ of sources we have as follows:

$$\delta_i(a_k, x_m, \varphi_i(a_k-1, \cdots, a_k-1, n, x_1, \ldots, x_n)) = a_k, \text{ where } a_k = \delta_i(a_k, (a_k-1 \oplus x_1), x_1),$$

if $(j, i) \in E$, and

$$a_k = \delta_i(a_k, x_m, \varphi_i(a_k-1, \cdots, a_k-1, x_1)), \text{ if } (i, j) \in E,$$

where $w_m = x_1 \cdots x_n \in \Sigma^n$ is the actual pseudo random string. Obviously, using the transition matrix of $A_i$, from $a_{k-1,1} \cdots a_{k-1,1} x_1, x_2$ we can determine $a_{k,i}$ for every $i \in V_1, (j, i) \in E$. Moreover, after calculating the values $a_i(i \in V_1)$, using the transition table of $A_i$, from $a_{k-1,1} \cdots a_{k-1,1} x_1, x_2$ we can determine $a_{k,i}$ for every $i \in V_2, (i, j) \in E$.

Then, concatenating the calculated blocks, we will get the ciphertext.

The decryption procedure is the following. Similarly as before, before the decryption procedure the pseudo random generator gets the first ciphertext block as its initialization vector $r_1 \cdots r_n \in \Sigma^n$.

Then the apparatus reads the ciphertext block-by-block, and after reading the next ciphertext block $c_1 \cdots c_n \in \Sigma^n$ (the first block first), it generates the second, the third and the further blocks of the plaintext in the following way.

The apparatus determines the state $a_{1} \cdots a_{n} \in \Sigma^n$ of key-automaton $B = (\Sigma^n, (\Sigma^n)^{2\log_2 n}, \delta_B)$ into which the automaton $B$ is taken from the state $a_{1} \cdots a_{n} \in \Sigma^n$ by the effect of $2\log_2 n$ consecutive strings in $\Sigma^n$ generated by the pseudo random generator.

Thus the pseudo random generator should generate a $2\log_2 n$ -long number of pseudo random sequences $w_1, \ldots, w_{2\log_2 n} \in \Sigma^n$ and going back from the last member $w_{2\log_2 n}$ to the first one $w_1$ the following procedure is performed.

Each of them takes the next temporal component (in opposite direction, i.e., the last one

4 Experimental results

The practical test was done using 16 byte (128 bit) long input blocks, output blocks and pseudo random blocks. First we present the size of the keyspace, then we continue our investigation with the test results of the the speed of the algorithm, and finally the effectiveness of the avalanche effect.

Using the above mentioned parameters with 256 possible states (1 byte long states) we need 16 automata having a transition matrix of $2^{256} = 65536$ lines and $2^3 = 256$ columns. Each cell of the automaton contains 1 byte long data (One state). The size of the matrix is 16 megabytes and the number of possible matrices is $256 \times 65536$, where the exclamation mark means the factorial operation. This protection is much more than good enough against brute-force
attacks. When we use isomorphic automata this huge number should be further increased to have $256^{165536} \times 256^{115} = 256^{60551}$ possible keys. Using the above mentioned parameters with half byte (4bits) long states, we need 32 automata having a transition matrix of $2^8 = 256$ lines and $2^4 = 16$ columns and each cell of the automaton contains half byte long data. In this case the size of the matrix is only 2 kilobytes and the number of possible matrices is $16^{256}$. Using permutation automata this can be increased to $16^{257}$ possible keys, which is still more than enough against brute-force attacks. However, we recommend the 8 bit version, because the number of calculations during the encoding and decoding process is less and the effectiveness of the avalanche effect is better.

The practical test of the encoding and decoding algorithm was done on an average desktop PC, (3.1 GHz Intel Core i3-2100 processor, 4 Gigabyte RAM). The program we used was a well written C# implementation. The results of the speed tests of the 8 bit version can be seen in Table 1.

The results of the speed tests show that using an average PC the encoding time is more than 4 megabytes per second, and decoding time is about the same.

The avalanche effect is a very important property of block ciphers. The block cipher is said to have avalanche effect when a small change in the plaintext block results in a significant change in the corresponding ciphertext block, further, a small change in the ciphertext block results in a significant change in the corresponding plaintext block. We tested the avalanche effect in the following way. We chose 1000000 random plaintext blocks, encoded them and then we changed 1 bit in each plaintext block, encoded again, then we calculated the number of different bytes in the ciphertext blocks pair-wise. We also tested the opposite case, namely, we chose 1000000 random ciphertext blocks, decoded them and then we changed 1 bit in each ciphertext block, decoded again and calculated the number of different bytes in each plaintext block pair-wise. During the first test we used just the first two rounds of encoding and decoding. The results can be seen in Table 2. When we change only one bit in the plaintext block the difference between the corresponding ciphertext blocks will be really huge in the majority of cases. The same effect can be seen in the opposite case: changing one bit in the ciphertext block results in a huge difference in the plaintext block as well. Although it was a good result, we also made a further test with the full 4-round algorithm. The results can be seen in Table 3.

Furthermore, we calculated the optimal avalanche effect. For this, we chose $2 \times 1000000$ completely random blocks and then calculated the difference between them pair-wise. The results are in Table 4.

We can assume that using the 8-bit version of the algorithm with 128 bit long blocks and 4 rounds the algorithm has the maximal avalanche effect and an appropriate speed (4 megabyte/s). Of course the speed of the algorithm depends on the hardware, the programming language and the actual program code as well.

5 The 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 pseudo random number generators. In case of each statistical test a set of P-values was produced. Given a significance level $\alpha$, if the P-value is less than or equal to $\alpha$ then test suggests that the observed data is inconsistent with our null hypothesis, i.e. the ‘hypothesis of randomness’, so we reject it. We used $\alpha = 0.01$ as it is common in such problems in cryptography. An $\alpha$ 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 P-value less than or equal to 0.01 would lead to the conclusion that the sequence is non-random.

One of the criteria used to evaluate the AES candidate algorithms was their demonstrated suitability as random number generators. 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. Randomness testing was 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 at 2$^{20}$ bits, 300 binary sequences, and $\alpha = 0.01$, respectively. Furthermore, Table 5 shows the length parameters we used.

In order to analyze the output of the algorithm we encrypted the Rockyou database, which contains more than 32 millions of cleartext passwords (see e.g: [Tihanyi et al., 2015]). Applying the NIST test for the encrypted file it has turned out that the output of the algorithm can not be distinguished in polynomial time from true random sources by statistical tests. The exact p-values of the evaluation of the ciphertext are shown in Table (6). We also tested the uniformity of the distribution of the p-values obtained by the statistical tests included in NIST, which is a usual requirement in the literature (see e.g. [Rukhin et al., 2010]). The uniformity of p-values provide no additional information about the type of the cryptosystem. 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. [Rukhin et al., 2010]).

6 Conclusions

The output of our crypto algorithm has passed all statistical tests of the NIST suite we performed and we were not
Table 1: Encoding and decoding speed test.

| Type of the plaintext | Size    | Encoding time | Decoding time | Encoded bytes per second |
|-----------------------|---------|---------------|---------------|--------------------------|
| Image: JPG            | 205216  | 00:00.0470960 | 00:00.0456500 | 4357397.6558519           |
| Document: PDF         | 204768  | 00:00.0459240 | 00:00.0454752 | 4458845.0483407           |
| Text: TXT             | 204848  | 00:00.0467470 | 00:00.0461294 | 4382056.6025627           |
| Compressed: RAR       | 204848  | 00:00.0471470 | 00:00.0454830 | 4344878.7833796           |
| Compressed: RAR       | 104883392 | 00:25.9539778 | 00:27.2784568 | 4041129.7569962           |
| Compressed: RAR       | 524613552 | 02:10.6843636 | 02:08.6140492 | 4014355.9454882           |
| Compressed: RAR       | 1102971104 | 04:28.121944 | 04:08.2624464 | 4442762.5683785           |

Table 2: Character differences after 2 rounds of encoding.

| different characters in one block | encoding | decoding |
|-----------------------------------|----------|----------|
| 0                                 | 0        | 0        |
| 1                                 | 0        | 0        |
| 2                                 | 1        | 1        |
| 3                                 | 0        | 0        |
| 4                                 | 36       | 40       |
| 5                                 | 3        | 1        |
| 6                                 | 72       | 89       |
| 7                                 | 125      | 136      |
| 8                                 | 5574     | 5594     |
| 9                                 | 11       | 4        |
| 10                                | 179      | 225      |
| 11                                | 410      | 396      |
| 12                                | 11050    | 11064    |
| 13                                | 880      | 921      |
| 14                                | 22670    | 22397    |
| 15                                | 43064    | 42710    |
| 16                                | 915924   | 916422   |

Table 3: Character differences after 4 rounds of encoding.

| different characters in one block | encoding | decoding |
|-----------------------------------|----------|----------|
| 0-12                              | 0        | 0        |
| 13                                | 37       | 28       |
| 14                                | 1717     | 1746     |
| 15                                | 59403    | 59145    |
| 16                                | 938842   | 939081   |

Table 4: Optimal avalanche effect.

| different characters in one block | 0-12 | 13 | 14 | 15 | 16 |
|-----------------------------------|------|----|----|----|----|
| 0-12                              | 0    |    |    |    |    |
| 13                                | 32   |    |    |    |    |
| 14                                | 1693 |    |    |    |    |
| 15                                | 58681|    |    |    |    |
| 16                                | 939594|    |    |    |    |
able to distinguish it from true random sources by statistical methods. Statistical analyses of a cryptosystem is a must-have requirement, and these test results are good indicators that further analyses can and should be done in order to check further properties. Cryptanalysis methods like chosen-plaintext, known-plaintext and related-key attack techniques will be used to prove or disprove the strength of the cryptosystem. These problems are the subject of our future research.

Many information systems such as computers and computer networks may be simulated by means of a queueing system. In general, queueing systems model is developed assuming the arrival rate and service intensity to be in the equilibrium state. The well-known methods of the queueing system investigation are based on the stationary behaviour of the input flow and service duration. Taking into account these characteristics as well as technical-economic criteria, the optimal system performance parameters are determined.

In real conditions the input flow arrival rate is affected by the step-by-step influence and the system state can essentially differ from the desired one. Here we come across the problem of compensating these differences with the purpose of equalizing the real value of output of customers’ flow to the desirable one.

The main idea of this work lies in the identification of the queueing system as the control object with further construction of discrete control closed scheme.

| Table 5: 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          |

7 Acknowledgments

This work was supported by the National Research, Development and Innovation Office of Hungary under Grant No. TET 16-1-2016-0193.

References

[Dömösi and Horváth, 2015a] Dömösi, P. and Horváth, G. (2015). A novel cryptosystem based on abstract automata and Latin cubes. Studia Scientiarum Mathematicarum Hungarica, 52(2):221–232. https://doi.org/10.1556/012.2015.52.2.1309

[Dömösi and Horváth, 2015b] Dömösi, P. and Horváth, G. (2015). A novel cryptosystem based on Gluškov product of automata. Acta Cybernetica, 22:359–371. https://doi.org/10.14232/actacyb.22.2.2015.8

[Dömösi et al., 2017] Dömösi, P., Gáll, J., Horváth, G and Tihanyi, N. (2017). Statistical Analysis of DH1 Cryptosystem. Acta Cybernetica, 23:371–378. https://doi.org/10.14232/actacyb.23.1.2017.20

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

[Mezenes and Vanstone, 1996] Menezes, P. C. O. A. J. and Vanstone, S. A. (1996). Handbook of Applied Cryptography ser. Discrete Mathematics and Its Applications. CRC Press. https://doi.org/10.1201/97814398718492

[Rukhin et al., 2010] Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S. (2010). NIST Special Publication 800-22: 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 August 2016. https://doi.org/10.6028/nist.sp.800-22

[Tihanyi et al., 2015] Tihanyi, N., Kovács, A., Vargha, G., Lénárt, A. Unrevealed Patterns in Password Databases Part One: Analyses of Cleartext Passwords. Technology and Practice of Passwords. PASSWORDS 2014. Lecture Notes in Computer Science, vol 9393. https://doi.org/10.1007/978-3-319-24192-0_6
| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | P-value | Proportion | Statistical Test |
|----|----|----|----|----|----|----|----|----|------|----------|-------------|------------------|
| 28 | 35 | 23 | 33 | 43 | 34 | 32 | 23 | 26 | 23   | 0.162606 | 296/300     | Frequency        |
| 25 | 29 | 35 | 38 | 27 | 23 | 26 | 27 | 31 | 39   | 0.407091 | 298/300     | BlockFrequency   |
| 28 | 37 | 26 | 37 | 32 | 28 | 25 | 36 | 25 | 26   | 0.574903 | 297/300     | CumulativeSums   |
| 26 | 30 | 31 | 30 | 33 | 27 | 24 | 38 | 28 | 33   | 0.840081 | 295/300     | CumulativeSums   |
| 33 | 20 | 33 | 26 | 32 | 28 | 44 | 25 | 30 | 29   | 0.205897 | 297/300     | Runs             |
| 23 | 33 | 40 | 24 | 31 | 22 | 31 | 29 | 38 | 29   | 0.284959 | 297/300     | LongestRun       |
| 24 | 28 | 40 | 32 | 24 | 30 | 30 | 27 | 37 | 28   | 0.527442 | 297/300     | Rank             |
| 34 | 35 | 23 | 33 | 30 | 35 | 27 | 34 | 23 | 26   | 0.623240 | 298/300     | FFT              |
| 35 | 31 | 30 |
|    | 29 | 30 | 29 | 32 | 28 | 23 | 33   | 0.958773 | 295/300     | NonOverlapping– | Template       |
|    |    |    |    |    |    |    |       |       |       |          |             |                  |
|    |    |    |    |    |    |    |       |       |       |          |             |                  |
| 25 | 27 | 25 | 29 | 40 | 39 | 29 | 33 | 26 | 27   | 0.419021 | 299/300     | OverlappingTemplate |
| 32 | 29 | 21 | 20 | 29 | 37 | 34 | 28 | 30 | 40   | 0.220031 | 298/300     | Universal        |
| 35 | 33 | 28 | 34 | 26 | 26 | 27 | 30 | 33 | 28   | 0.935716 | 299/300     | ApproximateEntropy |
| 21 | 17 | 24 | 23 | 15 | 15 | 18 | 12 | 15 | 17   | 0.516465 | 171/177     | RandomExcursions |
|    |    |    |    |    |    |    |       |       |       |          |             |                  |
|    |    |    |    |    |    |    |       |       |       |          |             |                  |
| 23 | 16 | 15 | 16 | 14 | 26 | 12 | 18 | 18 | 19   | 0.384836 | 172/177     | RandomExcursions– |
|    |    |    |    |    |    |    |       |       |       |          |             | Variant         |
|    |    |    |    |    |    |    |       |       |       |          |             |                  |
|    |    |    |    |    |    |    |       |       |       |          |             |                  |
| 23 | 27 | 38 | 25 | 27 | 43 | 41 | 24 | 24 | 28   | 0.042808 | 298/300     | Serial          |
| 28 | 28 | 25 | 45 | 32 | 32 | 33 | 28 | 25 | 0.253551 | 296/300     | Serial          |
| 32 | 25 | 33 | 40 | 20 | 31 | 35 | 15 | 35 | 0.039244 | 295/300     | LinearComplexity |
