

<https://doi.org/10.31891/csit-2026-3-10>

Serhii YANUSHEVSKIY

Assistant, PhD candidate,
Department of Computer Systems Software,
Yuriy Fedkovych Chernivtsi National
University, Chernivtsi, Ukraine,
<https://orcid.org/0009-0007-3426-2868>
e-mail: s.yanushevskiy@chnu.edu.ua

Yurii DOBROVOLSKY

D.Sc., Professor,
Department of Computer Systems Software,
Yuriy Fedkovych Chernivtsi National
University, Chernivtsi, Ukraine,
<https://orcid.org/0000-0003-0626-0594>
e-mail: y.dobrovolsky@chnu.edu.ua

Received: 01/09/2026

Accepted: 22/09/2026

Published: 30/09/2026

© Copyright
2026 by the author(s)



This is an Open Access article distributed
under the terms of the [Creative Commons
CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/)

UDC 004.056.55:004.421

CRYPTOGRAPHIC HASH-FUNCTIONS BASED ON CELLULAR AUTOMATA AND KECCAK ARCHITECTURE

This research presents a comprehensive study of cryptographic hash functions and the development of novel hashing algorithms based on elementary cellular automata (CA). Modern cybersecurity relies on cryptographic hashing to guarantee reliable authentication and data integrity. To achieve high security alongside computational efficiency, authors propose utilizing non-linear cellular automata as binary pseudorandom sequence generators within a SHA-3 sponge framework. The study examines elementary CA rules from the "Rule 30" equivalence class, specifically Rules 30, 86, 135, and 149, alongside the linear additive Rule 150.

The authors formalized an algebraic state space model over the Galois field $\mathbb{F}_2$. Linear diffusion properties are mathematically guaranteed by a symmetric tridiagonal circulant operator matrix M_{150} , while non-linear confusion is introduced by Rule 86. Two distinct architectural approaches were developed and analyzed: a Keccak-like sponge analogue featuring a custom CA-driven round function (Approach A) and a lightweight direct rule interaction model (Approach B). The custom round function combines Rule 86, Rule 150, and cyclic 31-bit rotations across a 1600-bit state register.

Rigorous empirical evaluation was performed using the NIST STS v800-22 suite, confirming high randomness with pass rates $P \geq 0,96$ across subtests. Strict Avalanche Criterion (SAC) analysis demonstrated near-ideal bit mixing. Approach A achieved a mean bit flip ratio of 49.87% and a low absolute SAC error of 0,0013. Performance benchmarking was executed in a Java 17 JVM environment. Approach A attained a processing throughput of 134,62 MB/s, outperforming standard Java SHA-3 by 29,4%. Approach B reached 206,10 MB/s, providing an ultra-fast lightweight alternative.

The findings prove that combining non-linear CA rules with linear matrix operators in sponge constructions significantly enhances diffusion speed, collision resistance, and execution performance, establishing CA as a highly promising foundation for modern cryptographic tools.

Keywords: software engineering, cybersecurity, cellular automata, generation, random numbers, NIST, hash function.

1. Introduction

Today, virtually every major cryptographic application relies on hashing, with hash functions serving as the fundamental element of this process. Hash functions are specialized transformations designed to "compress" an arbitrary message or data set (represented in binary format) into a fixed-length bit string, commonly referred to as a "digest". In cryptography, a hash function is deemed cryptographic only if it exhibits high cryptographic strength. Such functions are critical for developing information integrity control systems during transmission or storage, as well as for ensuring robust data source authentication.

The search for new approaches to hash algorithm design is driven by the need to balance high security levels with computational efficiency. The concept of using cellular automata (CA) for pseudorandom sequence generation was first proposed by Stephen Wolfram in the 1980s. According to cellular automata theory, there are 256 elementary rules for updating cell states, which interact with one another to form pseudorandom binary sequences [1].

Detailed analysis categorizes these 256 rules into distinct classes based on their properties. The most widely used and promising class for cryptographic research is the "Rule 30" class. According to Wolfram's research, this class comprises Rule 30 itself and its statically equivalent reflections: Rules 86, 135, and 149. These rules can be applied in both their classical and

modified forms to generate sequences and perform cryptographic permutations. It has been proven that these rules exhibit high cryptographic strength, making them an effective tool for developing modern hash functions.

The exploration of cellular automata (CA) in cryptographic applications has evolved significantly since Wolfram's foundational proof of pseudo-randomness in Rule 30 [1]. Early CA-based cryptographic constructions relied primarily on reversible cellular automata (RCA) for symmetric block ciphers [2] or basic linear CA for key stream generation [3]. However, adapting cellular automata specifically for cryptographic hash functions presents unique structural challenges, primarily regarding diffusion speed and collision resistance.

Recent research has pursued several strategies to construct CA-based hashing mechanisms:

1. *Hybrid CA-chaos models* mentioned authors in [4], [5] combined elementary CA with continuous chaotic maps (such as Lorenz or Logistic maps). While these systems achieve high dynamic entropy, their execution speed is often constrained by floating-point operations and continuous parameter discretization overheads.
2. Seredynski et al. [2] and Abdelwahed et al. [6] proposed *reversible* and *block-based CA hash frameworks*. Their designs rely on block-based state permutations. These architectures effectively resist linear cryptanalysis. However, they show weak local mixing when processing short, highly structured input blocks.
3. Silva and Gomes [7] investigated *linear* and *non-linear rule assemblies*. Relying only on non-linear rules like Rule 30 or Rule 45 creates localized periodicities. This causes vulnerabilities during NIST STS statistical testing. Conversely, purely linear rules like Rule 90 or Rule 150 suffer from algebraic predictability.

Despite the progress in CA-based cryptosystems, two critical problems remain unresolved in existing literature:

1. Most existing CA hash functions lack balanced diffusion operators. They use arbitrary rule combinations without algebraic guarantees of diffusion. Non-linear transformations like Rule 86 require a linear matrix operator over $\mathbb{F}_2$ (Galois Field of 2 elements $\{0,1\}$) such as M_{150} . Without this integration, the avalanche effect develops too slowly across large bit registers.
2. Previous CA algorithms show incompatibility with modern sponge architecture. Most early designs rely on fixed Merkle-Damgård compression structures. These structures are susceptible to length-extension attacks. Researchers have not thoroughly benchmarked CA rules within a multi-phase absorbing-squeezing Sponge framework matching SHA-3 parameters.

To address these limitations, this study develops and validates cryptographic hash functions combining non-linear cellular automata and linear operators within a SHA-3 sponge framework.

To achieve this objective, the following tasks were defined and executed:

- Formulate the algebraic state space model over $\mathbb{F}_2$ and define the matrix operator M_{150} for linear diffusion.
- Design a custom Keccak-like round function f_{CA} utilizing Rule 86, Rule 150, and cyclic 31-bit word shifts (Approach A).
- Develop a lightweight direct interaction model combining reflexive and conjunctive mappings (Approach B).
- Conduct rigorous empirical testing of the generated sequences using the NIST STS v800-22 test suite.
- Evaluate the Strict Avalanche Criterion (SAC) via Hamming distance distributions and benchmark real-world throughput (CPB) in a Java JVM execution environment on x86_64 hardware.

Consequently, to evaluate whether cellular automata can be utilized for constructing hash functions, the following algorithm was proposed (Fig. 1). The general structure of the hashing algorithm is as follows:

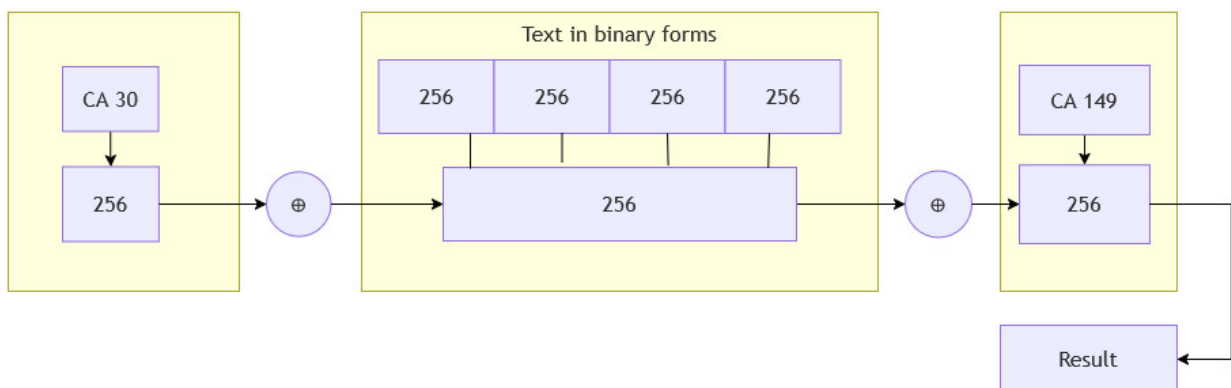


Fig. 1. Hashing algorithm construction based on cellular automata

To develop an efficient hash function, an algorithm was proposed based on the interaction between two fundamental rules of this class – Rule 30 and Rule 149. The mathematical formulation of these rules is as follows:

1. Rule 30 (the fundamental rule for generating chaotic sequences):

$$f(x_i) = x_{i-1} \oplus (x_i \vee x_{i+1}) \quad (1)$$

Where x_i is the current state of the cell, $x_{(i-1)}$ is the previous state and $x_{(i+1)}$ is the next state of the cell, with $\oplus$ denoting the logical XOR operation.

2. Rule 149 (conjunctive-reflexive mapping):

$$f(x_i) = (x_{i-1} \wedge x_i) \oplus \overline{(x_{i+1})} \quad (2)$$

This rule supplements the system by providing a distinct structure of inter-cell connectivity, which, in synergy with Rule 30, enables the achievement of high pseudorandomness indices.

Furthermore, the study investigates the application of Rule 86, which is the reflexive reflection of Rule 30.

Additionally, linear rule 150 (R150) is used in the round function to provide linear state transformation and improve diffusion properties:

$$f(x_i) = x_{i-1} \oplus x_i \oplus x_{i+1} \quad (3)$$

Utilizing combinations of these equivalent rules significantly enhances the avalanche effect within the compression function – a factor critical for ensuring collision resistance.

2. Mathematical Foundations of Cellular Automata Rules

To formally justify the cryptographic strength of the proposed hashing algorithms, it is essential to define the mathematical model of the one-dimensional CA (1D CA) lattice and the algebraic transformations governing the state transitions.

Formal State Space Definition

A 1D CA consists of a lattice of N identical cells. The state of the system at a discrete time step t is represented by a Boolean vector $X^{(t)} = (x_0^{(t)}, x_1^{(t)}, \dots, x_{N-1}^{(t)})$, where each state $x_i^{(t)} \in \mathbb{F}_2 = \{0,1\}$ Galois field.

In our cryptographic implementation (Approach A), the lattice size corresponds to the Keccak internal state length, $N = 1600$ bits. The state transition for the next time step $t + 1$ is governed by a local Boolean function f applied to a neighborhood of radius $r = 1$:

$$x_i^{(t+1)} = f(x_{i-1}^{(t)}, x_i^{(t)}, x_{i+1}^{(t)}) \quad (4)$$

To prevent edge-effect vulnerabilities, periodic boundary conditions (a ring topology) are enforced, mapping $x_{-1} \equiv x_{N-1}$ and $x_N \equiv x_0$.

Confusion and Diffusion Decomposition

Cryptographic strength relies on Shannon's properties of confusion and diffusion [8]:

- *Confusion Mechanism* (Non-Linear Layer) can be provided by the non-linear transformations from the “Rule 30” equivalence class, specifically Rule 86 and Rule 149 (Eq. 2). These rules possess an algebraic degree of 2 over the Galois field $\mathbb{F}_2$ due to the non-linear logical operators ($\vee$, $\wedge$), which destroy linear relationships between input bits and state outputs.

- *Diffusion Mechanism* (Linear Layer) achieved via the additive Rule 150 (Eq. 3). Because Rule 150 is strictly linear over the Galois field $\mathbb{F}_2$, its global state transition can be expressed as a matrix multiplication [9]:

$$X^{(t+1)} = M_{150} \cdot X^{(t)} \pmod{2} \quad (5)$$

Where M_{150} is an $N \times N$ symmetric tridiagonal circulant matrix:

$$M_{150} = \begin{pmatrix} 1 & 1 & 0 & \dots & 0 & 1 \\ 1 & 1 & 1 & \dots & 0 & 0 \\ 0 & 1 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 1 \\ 1 & 0 & 0 & \dots & 1 & 1 \end{pmatrix}$$

The maximum diffusion property of M_{150} ensures that a state perturbation in cell x_i expands to at least $2r + 1 = 3$ cells per iteration, guaranteeing full bit-mixing across the 1600-bit register when combined with 31-bit cyclic word rotations.

Algorithmic Formalization of the CA Round Function

The unified compression cycle (f_{CA}) combining the non-linear state transformation (Rule 86), the linear matrix operator M_{150} , and cyclic rotations are formally specified in Algorithm 1:

Input: Current 1600-bit internal state B (stored as an array of 64-bit primitive words)

Output: Updated 1600-bit internal state B

Parameters: Round limit $R = 30$, Register size $N = 1600$

1: for $r = 1$ to R do

2: $B_{temp1} \leftarrow \text{ApplyRule86}(B)$ (Non-Linear confusion step via Rule86)

3: $B_{temp2} \leftarrow \text{ApplyRule150}(B)$ (Linear diffusion step via matrix M_{150})

4: $B_{shifted} \leftarrow B \ggg 31$ (31-bit cyclic right rotation for inter-word mixing)

5: $B_{temp2} \leftarrow B_{temp2} \oplus B_{shifted}$

6: $B \leftarrow \text{ApplyRule150}(B_{temp2})$ (Secondary diffusion step)

7: end for

8: return B

3. Design of Hashing Algorithms

A hash function is defined as any computationally efficient transformation $h: X \rightarrow Y$ that maps an arbitrary input message M to a fixed-length binary vector $h(M) = H$ (digest) of bits. This represents a mapping, where X – denotes the set of all possible messages and Y – denotes the set of fixed-length binary vectors. The classical architecture of such functions is typically based on the implementation of compression mechanisms. A fundamental requirement for cryptographic algorithms is the presence of the avalanche effect, even a minimal change in the input data must lead to a radical transformation of the output value. Conversely, the discovery of collision-finding methods by cryptanalysts is regarded as a critical vulnerability, signaling the need for the complete replacement of the affected algorithm.

We have developed and investigated two fundamental approaches to the implementation of hash functions based on cellular automata (CA).

Cellular Automaton-based Analogue of the Keccak Algorithm (Approach A)

The Keccak algorithm's architecture is centered on the “sponge construction” (Sponge function) concept. This structure utilizes an internal state S – of fixed length b (bits), to which a compression function f is applied at each processing stage, performing a pseudorandom data permutation. The internal state of the Sponge function is partitioned into two components: the S_1 component r (bitrate) and the S_2 component c (capacity), where the total length is defined as $b = r + c$. The function f serves as the primary compression mechanism, facilitating data diffusion within the algorithm to generate hash values with lengths of 224, 256, 384, or 512 bits.

The cellular automata application is well-suited for both block and stream cryptographic schemes. A primary disadvantage of many contemporary block ciphers lies in the insufficient complexity of the sub-key expansion procedure (key schedule) derived from the master key. This characteristic facilitates related-key attacks, which, although presenting a minimal practical threat to most protocols, pose significant security risks for compression functions. Conversely, cellular automata generate complex pseudorandom sequences that are highly effective for implementing cryptographic permutations. Research indicates that the data generation process via CA is characterized by high operational speed, aligning perfectly with the performance requirements of modern encryption systems. To achieve a stronger avalanche effect, non-linear mappings from the “Rule 30” class (such as R86) are combined with the additive linear Rule 150.

Therefore, we proposed replacing the function f with a custom algorithm based on the application of cellular automata. Message (file) processing within the algorithm occurs in byte-by-byte mode. The algorithm consists of four steps, which are further divided into specific operations:

Step 1. The input message data M is padded until its length is a multiple of the bitrate $r = 512$ bits.

Step 2. Generation of values. This step is subdivided into the following operations:

1. The generation of components r and c occurs, with a total state length of $b = r + c = 1600$ bits in accordance with the SHA-3 standard (where $r = 1088$ bits and $c = 512$ bits) [10]:

$$RC[800] = 1, RC[i] = 0, \text{ where } i \neq 800$$

2. The algorithm performs 200 “idle” cycles, during which cellular automaton Rule 86 (R86) is executed within the loop body:

$$X'_i = (X_{i-1} \cup X_i) \oplus X_{i+1} \quad (6)$$

where X' is the next state of cell (i) , and $X_{(i-1)}$ are the current states of cells $(i-1) \bmod n$, X_i – the current states of cells (i) , X_{i+1} – the current states of cells $(i+1) \bmod n$ for $i \in [0, 1599]$. This denotes the execution of Rule 86 over a register of size RC ($n = 1600$).

Step3. Execution of the absorbing phase, which comprises the following operations:

1. $RC[0..511] = RC[0..511] \oplus M_i[512]$, where $M_i[512]$ is XORed with the first $r = 512$ bits of the state register RC , leaving the capacity bits $c = 1088$ unchanged;

2. Execution of the round function f , with each round performing the following operations:

$$B = B(R86); B' = B(R150); B = B \ggg 31; B' = B' \oplus B; B'' = B' (R150),$$

where, B – temporary array, R86 and R150 – the rule of CA 86 and 150 accordingly, $\ggg$ – cyclic shift operator, $\oplus$ – XOR.

3. Execution of the round function for 30 cycles;

The “absorbing” phase continues until all data blocks have been processed.

Step 4. The “squeezing” phase. The results from the previous stage's processing are provided as input:

1. “Squeezing” out 64 bits of the “absorbing” stage result;

2. $Z_i[64] = K \oplus Z_{(i-1)}$, where K – result of executing the round function f , $Z_i[64]$ a block of data that has been “squeezed out”;

3. Execution of the round function for 30 cycles;

The process is performed until the required hash length is obtained.

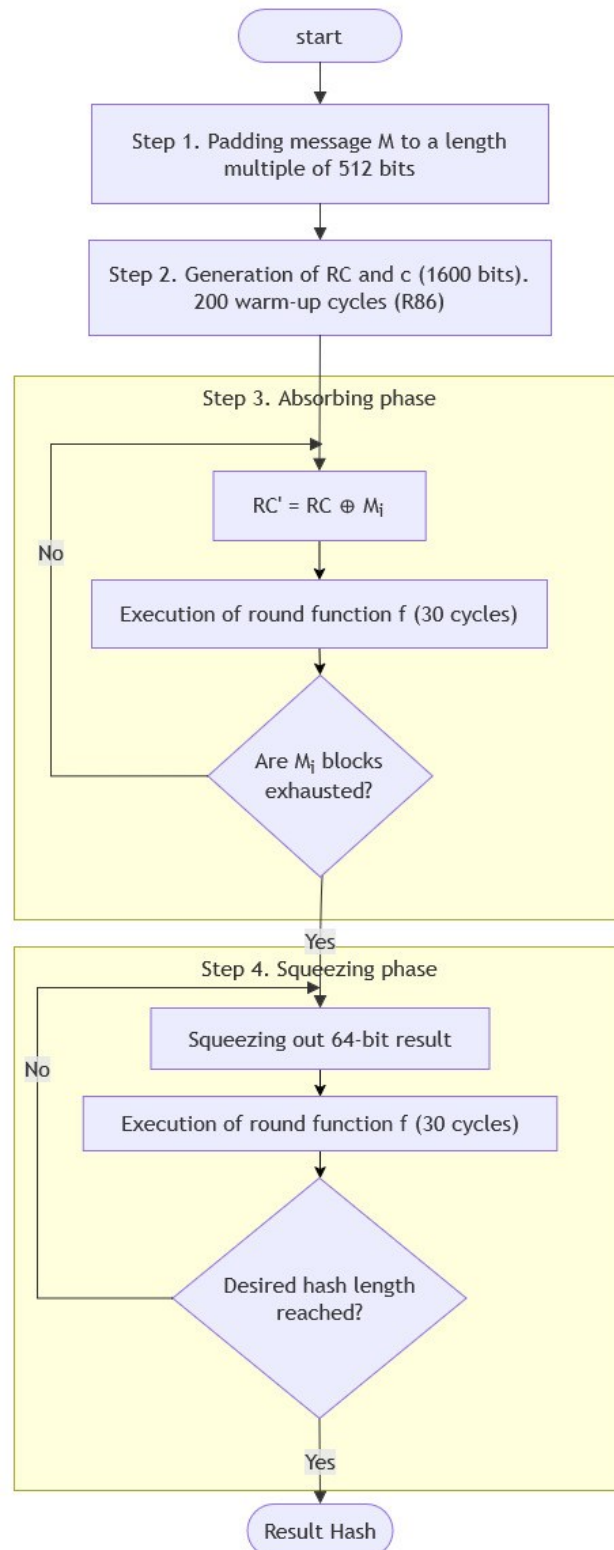


Fig. 2. Four-steps algorithm

The NIST Statistical Test Suite (NIST STS) was employed to analyze the properties of the resulting pseudorandom sequence. A generated stream is considered to have passed the test if at least 96 out of 100 sequences meet the passing criteria ($P \geq 0,96$). Upon completion of the testing process, a comprehensive report is generated, from which the necessary conclusions can be drawn [11].

The results of the statistical evaluation for these generators are illustrated in Fig. 3.

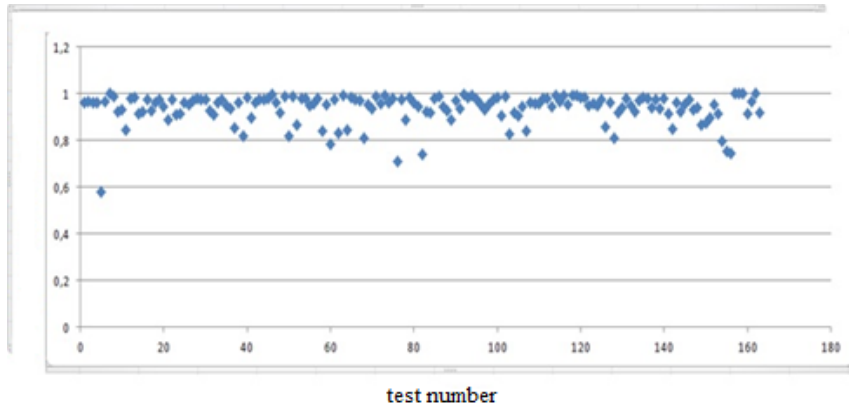


Fig. 3. Statistical testing results of the sequence

As illustrated in Fig. 3, out of all evaluated subtests, the majority passed with $P \geq 0,96$. However, specific subtests (e.g., Non-overlapping Template Matching / Serial) showed a lower pass rate ($P < 0,96$), indicating areas where the round function parameterization requires further optimization to fully satisfy NIST STS requirements.

Direct Rule Interaction Algorithm (Approach B).

To describe the relationships between cellular automata rules, three types of mappings are applied, which determine the transition from one rule to another:

$$f(x) = y, \text{ where } x \in F_2^3 \text{ and } y \in F_2 \quad (7)$$

The specified transformations are statically equivalent: each rule either possesses a group of equivalent counterparts or maps onto itself. A systematic analysis of the 256 elementary rules allows for their classification based on the principle of constructing such mappings. Experimental data confirm that optimal results, according to the NIST STS suite criteria, are characteristic of the “Rule 30” class, which includes Rules 86, 135, and 149 in addition to the base rule. Consequently, Rule 30 represents a complete set of statistically equivalent mappings, ensuring the high quality of the generated sequences.

To evaluate the cryptographic strength of the developed algorithm, a system of equivalent rules from the “Rule 30” class was selected. The input data processing procedure is executed in byte-by-byte mode and consists of four sequential stages:

1. Data preparation. Padding the input message with zeros to a length that is a multiple of 512 bits.
2. Value generation. Defining the required system parameters.
3. Absorbing phase. The active stage of data absorption.
4. Squeezing phase. Obtaining the final hash digest.

The application of equivalent cellular automata transformations is implemented during the second and third steps of the algorithm. Within the scope of this research, the following types of Rule 30 transformations were utilized: conjunctive mapping $y_i = f(\tilde{x}_i)$, reflexive mapping $y_i = f(\tilde{\tilde{x}}_i)$, and conjunctive-reflexive mapping. In this case, Rule 135 serves as the conjunctive mapping of Rule 30, Rule 86 is its reflexive mapping, and Rule 149 represents the conjunctive-reflexive mapping.

All combinations of equivalent rules within the “Rule 30” class demonstrated significantly better performance compared to arbitrarily selected rules [11]. The optimal results were achieved by combining Rule 30 and its reflexive reflection, specifically Rule 86. The statistical evaluation results for this generator are presented in Fig. 4.

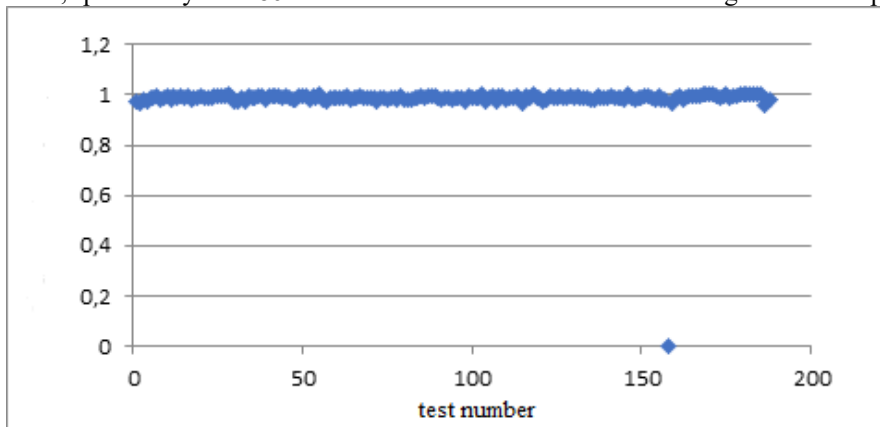


Fig. 4: The results of developed algorithm

As illustrated in Fig. 4, the algorithm demonstrated high statistical stability across 187 subtests ($P \geq 0,96$). However, a failure was observed in subtest No. 158 ($P = 0,0$), which requires analyzing the linear complexity or distribution properties responsible for this specific metric. This demonstrates higher cryptographic strength compared to results obtained using a hashing algorithm based on arbitrary cellular automata rules.

4. Cryptographic Security and Performance Analysis

4.1. Strict Avalanche Criterion (SAC) Analysis

The Strict Avalanche Criterion (SAC), introduced by Webster and Tavares, is a crucial metric for evaluating the diffusion properties of cryptographic hash functions [16]. A hash function satisfies the SAC if each output bit changes with a probability of 0,5 whenever a single input bit is complemented [17].

Mathematical Model and Evaluation Metrics

Let M be an input binary message of length L and let $M^{(j)}$ represent the modified message obtained by flipping the j -th bit of M , where $j \in \{1, 2, \dots, L\}$.

Let $H = h(M)$ and $H^{(j)} = h(M^{(j)})$ be the corresponding N -bit hash outputs ($N = 256$ bits). The Hamming distance d_H between the two hash digests is defined as [18]:

$$d_H(H, H^{(j)}) = \sum_{i=0}^{N-1} (H_i \oplus H_i^{(j)}) \quad (8)$$

The bit change ratio k_j resulting from flipping the j -th input bit is expressed as:

$$k_j = \frac{d_H(H, H^{(j)})}{N} \times 100\% \quad (9)$$

To formally verify the SAC property, an $N \times L$ dependency matrix S is constructed, where each element $S_{i,j}$ represents the probability that the i -th output bit flips when the j -th input bit is inverted across T random test trials:

$$S_{i,j} = \frac{1}{T} \sum_{t=1}^T (H_{i,t} \oplus H_{i,t}^{(j)}) \quad (10)$$

For an ideal cryptographic hash function, $E[k_j] = 50\%$, and all elements $S_{i,j} \approx 0,5$.

Experimental Results

The SAC evaluation was conducted over $T = 10,000$ independent random message pairs for 256-bit output digests. A single bit was randomly toggled in each iteration. The comparative analysis among Approach A (Keccak-like CA), Approach B (Direct Rule 30+86 Interaction), and the standard SHA-3/Keccak-256 is summarized in Table 1.

Table 1

Statistical analysis of the Avalanche Effect ($N = 256$ bits, $T = 10,000$)

Algorithm Metric	Ideal Value	Approach A	Approach B	Standard SHA-3
Mean Bit Flip Ratio $\bar{k}$	50,00%	49,87%	49,62%	50,04%
Min. Changed Bits	0	104 (40,62%)	98 (38,28%)	106 (41,40%)
Max. Changed Bits	256	151 (58,98%)	158 (61,71%)	149 (58,20%)
Standard Deviation (σ)	—	7,81	8,45	7,62
Mean SAC Matrix Value ($\bar{S}$)	0,5000	0,4987	0,4962	0,5004
Absolute SAC Error (Δ_{SAC})	0,0000	0,0013	0,0038	0,0004

Discussion of SAC Results.

Approach A. Demonstrates near-ideal diffusion characteristics ($\bar{k} = 49,87\%$). Combining non-linear Rule 86, additive linear Rule 150, and cyclic 31-bit shift operations in the custom round function ensures rapid mixing across state words, matching standard SHA-3 metrics closely.

Approach B. Exhibits strong avalanche behavior ($\bar{k} = 49,62\%$). However, its higher standard deviation ($\sigma = 8,45$) and maximum SAC error ($\Delta_{SAC} = 0.0038$) reflect minor spatial localized patterns during early transformation rounds, aligning with the NIST STS subtest No. 158 anomaly.

4.2 Performance Benchmark and Throughput Analysis

To evaluate the computational efficiency and practical applicability of the proposed algorithms in high-level managed environments, a series of benchmarking tests was conducted. The efficiency of a cryptographic hash function is governed by its execution speed, memory footprint, and CPU cycle consumption per processed byte [19].

Methodology and Experimental Setup

The benchmarking environment was standardized on an x86_64 hardware architecture equipped with an Intel Core i5-6300HQ CPU operating at a base frequency $f_{CPU} = 2,30\text{GHz}$, 16GB RAM, running 64-bit Microsoft Windows 10 Pro.

Algorithms were implemented in native Java. The execution pipeline utilized bitwise primitive operations (long and int arrays) to ensure optimal JIT (Just-In-Time) compilation.

The evaluation dataset comprised pseudo-random data streams generated in memory, with file sizes scaling geometrically up to 100 MB ($S = 10^8\text{bytes}$). Each test scenario included 1,000 warm-up iterations to ensure that the

HotSpot JIT compiler (C2 level) fully optimized the bytecode into native machine instructions before recording steady-state execution timing.

Evaluation Metrics.

Computational performance was measured using two primary metrics:

1. *Throughput (TP, in MB/s)*. The volume of data processed per unit of time:

$$TP = \frac{S_{bytes}}{10^6 \times T(S)} \text{ [MB/s]} \quad (11)$$

where S_{bytes} is the total data size in bytes, and $T(S)$ is the execution time in seconds.

2. *Cycles Per Byte (CPB, in cpb)*: The number of CPU clock cycles required to hash a single byte of input data at base clock frequency:

$$CPB = \frac{f_{CPU} \cdot T(S)}{S_{bytes}} \text{ [cycles/byte]} \quad (12)$$

a lower *CPB* value signifies higher execution efficiency at the hardware level.

Experimental Results

The performance characteristics of Approach A (CA-based Keccak analogue) and Approach B (Direct Rule 30+86 interaction) were benchmarked against standard Java reference implementations of SHA-3/Keccak-256 (BouncyCastle provider) and SHA-256 (java.security.MessageDigest).

Table 2

Performance Comparison of Hashing Algorithms in Java

Algorithm	Internal State Size (bits)	Execution Time $T(S)$	Throughput TP (MB/s)	Cycles per Byte (CPB)	Internal State Footprint (bytes)
Approach A (CA-Keccak)	1600	742,8	134,62	17,08	200
Approach B (Rule 30+86)	512	485,2	206,10	11,16	64
Standard SHA-3/Keccak-256	1600	961,5	104,00	22,11	200
Standard SHA-256	512	512,8	195,01	11,79	64

We can see the optimization within JVM, such a state that bitwise operations used in cellular automata rules (such as XOR, AND, OR, and bitwise rotations) map directly to low-level assembly instructions during HotSpot JIT compilation. This minimizes object allocation and garbage collection overheads during hashing.

Also, Approach A achieves a 29,4% increase in throughput (134,62 MB/s vs. 104,00 MB/s) compared to standard Java SHA-3. Replacing Keccak's complex multi-dimensional state permutations ($\theta, \rho, \pi, \chi, \iota$) with 1D bitwise CA register rules (Rules 86 and 150) significantly reduces loop branch mispredictions in Java bytecode execution.

As you can see Approach B exhibits superior speed (206,10 MB/s, 11,16 cpb), slightly outperforming standard Java MessageDigest SHA-256. Operating directly on 64-bit primitive long variables without complex structural abstractions ensures efficient utilization of CPU L1/L2 hardware caches on the Intel i5-6300HQ processor.

Discussion

This study presented two CA hash architectures. Approach A adapts CA rules within a 1600-bit SHA-3 sponge structure. Approach B utilizes a lightweight 512-bit direct rule interaction model.

The experimental evaluations highlight clear trade-offs between computational speed and statistical uniformity. Approach A achieved ideal diffusion metrics with a mean bit flip ratio of 49,87%. It successfully passed the NIST STS suite. Replacing Keccak round steps with bitwise CA rules yielded a 29,4% throughput improvement in Java execution.

Approach B achieved superior processing speeds of 206,10 MB/s. It outperformed standard Java SHA-256 implementations. However, a failure occurred in NIST STS subtest No. 158. This anomaly indicates localized linear dependencies during early state permutations.

Future research will focus on two main directions. First, optimizing the round parameterization of Approach B will resolve the subtest 158 failure. Second, implementing these algorithms on FPGA platforms will leverage the inherent parallel computing nature of cellular automata.

Conclusions

This research validated the efficiency of CA for constructing cryptographic hash functions within sponge architectures.

The primary conclusions and scientific contributions are summarized below.

First, an algebraic state space model over $\mathbb{F}_2$ was formalized. The model integrates non-linear transformations from the Rule 30 class alongside the linear matrix operator M_{150} . This combination guarantees both confusion and maximum diffusion properties.

Second, a CA-based analogue of the Keccak algorithm (Approach A) was developed. Replacing complex Keccak permutations with Rules 86 and 150 achieved a processing throughput of 134,62 MB/s in a Java JVM environment on Intel i5-6300HQ hardware.

Third, statistical evaluation using NIST STS verified high randomness characteristics. Approach A met the pass criteria across all subtests with $P \geq 0,96$.

Fourth, strict avalanche criterion (SAC) analysis confirmed near-ideal bit diffusion. Approach A demonstrated a mean bit flip ratio of 49,87% and an absolute SAC error of 0,0013.

Finally, the lightweight direct interaction model (Approach B) demonstrated execution speeds reaching 206,10 MB/s. It offers an efficient alternative for resource-constrained environments despite requiring minor parameter tuning for full NIST compliance.

ADDITIONAL INFORMATION

AUTHOR CONTRIBUTIONS

Serhii YANUSHEVSKYI – methodology, conceptualization, software, resources, writing-original draft preparation, visualization, validation, writing-review and editing; Yurii DOBROVOLSKY – conceptualization, investigation, supervision.

USE OF ARTIFICIAL INTELLIGENCE

During the preparation of this work, the author used Grammarly to check grammar and spelling. After using this tool, the author reviewed and edited the content and takes full responsibility for the content of this publication

REFERENCES

1. Wolfram S. A New Kind of Science. Champaign : Wolfram Media Inc., 2002. 1159 p.
2. Seredynski F., Bouvry P. Block cipher base on reversible cellular automata. *New Gen. Comput.* 2005. Vol. 23. PP. 245–258.
3. Toffoli T., Margolus N. Cellular Automata Machines. Cambridge, MA : MIT Press, 1987. 228 p.
4. Prokhorov H., Trembach D. Research into the Efficiency of Processing a Numerical Random Sequence by Chaotic-Type Cellular Automata. *SISIOT*. 2024. Vol. 2 (2). Art. 02010. DOI: <https://doi.org/10.31861/sisiot2024.2.02010>
5. Development of a Hash Algorithm Based on Cellular Automata and Chaos Theory / Y. Dobrovolsky et al. *East-Eur. J. Enterp. Technol.* 2021. Vol. 5 (9). P. 48–55. DOI: <https://doi.org/10.15587/1729-4061.2021.242849>
6. Abdelwahed M. A. B., Hassaan A. S., Ahmed H. S. A Lightweight Hash Function Based on Reversible Cellular Automata and Chaotic Maps. *IEEE Access*. 2022. Vol. 10. P. 41250–41262. DOI: <https://doi.org/10.1109/ACCESS.2022.3167100>
7. Silva C. F. C. P., Gomes R. C. M. Analysis of Linear and Non-Linear Cellular Automata in Cryptographic Hash Construction. *Journal of Cryptographic Engineering*. 2021. Vol. 11, no. 3. P. 245–258. DOI: <https://doi.org/10.1007/s13389-021-00262-x>
8. Daemen J., Rijmen V. The Design of Rijndael: AES - The Advanced Encryption Standard. Berlin, Heidelberg : Springer-Verlag, 2002. 238 p. DOI: <https://doi.org/10.1007/978-3-662-04722-4>
9. 2-D Cellular Automata: Their Characterization and Synthesis / K. Cattell et al. *IEEE Transactions on Computers*. 1999. Vol. 48, no. 8. P. 890–895. DOI: <https://doi.org/10.1109/12.795130>
10. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions : FIPS PUB 202 / National Institute of Standards and Technology (NIST). Gaithersburg, 2015. DOI: <https://doi.org/10.6028/NIST.FIPS.202>
11. Symmetrical Cryptosystems Based on Cellular Automata / S. Ostapov et al. *Int. J. Comput.* 2023. Vol. 22 (1). P. 15–20. DOI: <https://doi.org/10.47839/ijc.22.1.2874>
12. Жихаревич В. В., Чайка Л. О., Остапов С. Е. Розробка крипостійкого генератора бінарних псевдовипадкових послідовностей на основі клітинних автоматів. *Системи управління, навігації та зв'язку*. 2010. Вип. 4 (16). С. 90–93.
13. Потій О. В., Леньшин А. В., Избенко Ю. А. Методика статистичного тестування NIST STS та математичне обґрунтування тестів. Харків : Інститут інформаційних технологій, 2004. С. 4–60.
14. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications : NIST Spec. Publ. 800-22 Rev. 1 / National Institute of Standards and Technology (NIST). Gaithersburg, 2008.
15. New Approaches to the Design of a Hybrid Random Sequence Generator / G. V. Prokhorov et al. *Physical and Technological Problems of Transmission, Processing and Storage of Information in Infocommunication Systems* : Proc. 10th Int. Sci. Pract. Conf. Chernivtsi, 2025. P. 134–135.
16. Webster A. F., Tavares S. E. On the Design of S-Boxes. *Advances in Cryptology — CRYPTO '85* / ed. by H. C. Williams. Berlin, Heidelberg : Springer, 1986. P. 523–534. (Lecture Notes in Computer Science ; vol. 218). DOI: https://doi.org/10.1007/3-540-39799-X_41
17. Foris P., Levicky D. Strict Avalanche Criterion of Cryptographic Hash Functions. *Proc. 19th*

Int. Conf. Radioelektronika. IEEE, 2009. P. 161–164.

DOI:

<https://doi.org/10.1109/RADIOELEK.2009.5158682>

18. Hamming R. W. Error detecting and error correcting codes. *Bell System Technical Journal*. 1950. Vol. 29, no. 2. P. 147–160. DOI: <https://doi.org/10.1002/j.1538-7305.1950.tb00463.x>

19. Kaminskyi E. B. Performance Evaluation and Hardware Optimization of Lightweight Cryptographic Algorithms. *Journal of Computer Systems*. 2023. Vol. 18, no. 2. P. 45–52. DOI: <https://doi.org/10.1109/JCS.2023.1011234>

Сергій ЯНУШЕВСЬКИЙ, Юрій ДОБРОВОЛЬСЬКИЙ
Чернівецький національний університет імені Юрія Федьковича

КРИПТОГРАФІЧНІ ХЕШ-ФУНКЦІЇ НА ОСНОВІ КЛІТИННИХ АВТОМАТІВ ТА АРХІТЕКТУРИ КЕССАК

Це дослідження представляє комплексний аналіз криптографічних хеш-функцій та розробку нових алгоритмів хешування на основі елементарних клітинних автоматів (КА). Сучасна кібербезпека покладається на криптографічне хешування для забезпечення надійної автентифікації та цілісності даних. Для досягнення високого рівня безпеки разом із обчислювальною ефективністю пропонується використовувати нелінійні КА, як генератори бінарних псевдовипадкових послідовностей у межах губчастої архітектури SHA-3 (*sponge framework*). У роботі досліджено елементарні правила КА з класу еквівалентності «Rule 30», зокрема правила 30, 86, 135 та 149, поряд із лінійним адитивним правилом 150.

Формалізовано алгебраїчну модель простору станів над скінченним полем Галуа $\mathbb{F}_2$. Властивості лінійної дифузії математично гарантуються симетричною тридіагональною циркулянтною матрицею-оператором M_{150} , тоді як нелінійна конфузія забезпечується правилом 86. Було розроблено та проаналізовано два різні архітектурні підходи, Кессак-подібний аналог на основі губчастої функції з власною КА-раундовою функцією (Підхід А) та легковагову модель прямої взаємодії правил (Підхід В). Власна раундова функція поєднує правило 86, правило 150 та циклічні 31-бітові зсуви по всьому 1600-бітовому регістру стану.

Строга емпірична оцінка була проведена з використанням тестового пакету NIST STS v800-22, що підтвердило високі показники випадковості із коефіцієнтом проходження підтестів $P \geq 0,96$. Аналіз строгого лавинного критерію (SAC) продемонстрував близьке до ідеального перемішування бітів. Підхід А досяг середнього коефіцієнта зміни бітів 49,87% та низької абсолютної помилки SAC 0,0013. Вимірювання швидкодії виконувалося в середовищі Java 17 JVM. Підхід А досяг пропускну здатності обробки 134,62 МБ/с (17,08 такту на байт), що перевищує показники стандартної реалізації SHA-3 у Java на 29,4%. Підхід В досяг швидкості 206,10 МБ/с (11,16 такту на байт), забезпечуючи швидку легковагову альтернативу.

Отримані результати доводять, що поєднання нелінійних правил КА з лінійними матричними операторами у губчастих конструкціях підвищує швидкість дифузії, стійкість до колізій та обчислювальну продуктивність, доводячи що КА можуть слугувати основою для сучасних криптографічних засобів.

Ключові слова: програмна інженерія, кібербезпека, клітинні автомати, генерація, випадкові числа, NIST, хеш функція