Hindawi
Security and Communication Networks
Volume 2021, Article ID 6640052, 21 pages
https://doi.org/10.1155/2021/6640052

*Research Article*

# CFA: A New Family of Hybrid CA-Based PRNGs

**Fatima Ezzahra Ziani, Anas Sadak, Charifa Hanin, Bouchra Echandouri, and Fouzia Omary**

*Computer Science Department at the Faculty of Sciences, Mohammed V University, Rabat 1014, Morocco*

Correspondence should be addressed to Fatima Ezzahra Ziani; fatimaezzahra_ziani2@um5.ac.ma

Received 5 October 2020; Revised 9 May 2021; Accepted 12 May 2021; Published 28 May 2021

Academic Editor: Mamoun Alazab



In this article, a new Pseudorandom Number Generator (PRNG) construction is proposed. It is based on cellular automata (CAs) and comprises other cryptographic primitives organized as blocks. Each of these blocks has a purpose and serves toward obtaining a higher level of randomness. The construction described is modular, and each of its blocks can be replaced, modified, or adapted according to the user, the application, or the level of randomness required. The authors first describe a general structure and the design principles behind each of the components. Next, a concrete example using the SHA-3 hash function, a hybrid cellular automaton, and the AES block cipher is provided. Then, the security analysis and the statistical properties for this specific instance of the scheme are presented.

## 1. Introduction

Designing cryptographic primitives that achieve security strength and satisfy all of the cryptographic properties is a hard task. First, the development of cryptanalysis techniques makes it difficult for designers to build new primitives with a high security level. In addition, some cryptographic properties are conflicting like, for example, nonlinearity and correlation [1]. Therefore, a compromise between cryptographic properties should be found according to the security level the designer wants to achieve.

Many cryptographic applications make use of random numbers. Examples are session keys, seeds, salts, initial vectors, nonce, etc. However, for costs reasons, not all these applications use truly random bits generated from physical sources. Generally, a pseudorandom number generator is used. This cryptographic mechanism tries to generate sequences that are hard to distinguish from truly random sequences.

The motivation of this work is to design a versatile, portable, and secure PRNG family based on cellular automata. Consequently, it could be adapted easily according to the different applications requiring random sequences. The design presented here is modular in the sense that all the building blocks can be modified by the system's designer to meet specific requirements. Those building blocks are a hash function, a cellular automaton, and a block cipher. The mechanism of seeding and reseeding is also part of the design and ensures that the PRNG starting state is unpredictable. All those components were included to achieve a high degree of randomness, a high security level, and good cryptographic properties. Statistical tests, properties evaluation, and a selection procedure for the ruleset of the cellular automaton have been conducted to show that the generator proposed is a good candidate for generating high quality random sequences. The family of generators presented in this work can be used as standalone schemes to generate random sequences or as part of a bigger system, for example, within a stream cipher.

The previous CA-based PRNGs, found in [2–8], based the security of their systems on only cellular automata. Consequently, their designs cannot prevent some attacks like chosen-input attacks, direct cryptanalytic attacks, and side-channel attacks. They focused more on the randomness property, which made the systems lack the security level required by a secure PRNG. Instead, CFA, the PRNGs family

proposed and studied in this article, was designed by considering the randomness properties and known attacks to provide a high-security level. The main contributions of this work were to define a general design satisfying the requirements of a Cryptographically Secure PRNG, then the choice of the building blocks of this design. The construction of the cellular automaton ruleset provides good cryptographic properties (see Appendix A), which enhances the security of the PRNG in addition to the randomness properties. Secondly, the choice of the hash function and the block cipher are adequate for the proposed implementation. In addition, the seed file, the reseed, and the reseed trigger mechanisms make attacks trying to find out the initial configuration of the PRNG challenging.

The rest of this article is organized as follows. In Section 2, some basic notions about randomness, entropy, and cellular automata are defined. In Section 3, a brief description of related works is given. Next, in Section 4, the general scheme of the generator and a specific implementation are described. The experimental results and an analysis of that specific implementation follow in Section 5. Finally, Section 6 presents challenges, possible solutions, and future directions, followed by conclusions.

## 2. Preliminaries

*2.1. Randomness.* Random numbers are an essential component of cryptographic applications such as encryption, key generation, or content masking. The notions of random, randomness, and random numbers are often associated with unpredictability and uniform distribution.

A Random Number Generator (RNG) is an algorithm that produces sequences of numbers or bits that seem to be random, from a seed or from a continuously fed input. True Radom Number Generators (TRNGs) are a class of RNGs that use physical sources to generate "true" random sequences. However, working with truly random sequences has drawbacks like the reliability of the physical sources, the availability of the data, the limited amount of data, or the high cost of getting the data. Pseudorandom Number Generators (PRNGs) are another class of RNGs that use algorithms to generate random sequences as indistinguishable as possible from "truly" random sequences [9].

In general, an RNG must meet specific conditions. Its output should be independent, uniformly distributed, and unpredictable [9]. The generator should be efficient in terms of the amount of random numbers generated. In order to be used in a cryptographic environment, an RNG should additionally be resistant to well-known attacks [9]. This means that, despite having information about the input or the current/previous output of the generator, an attacker cannot guess its output (previous, current, or future).

*2.2. Entropy.* Entropy is another term associated with RNGs. The concept was first introduced in 1948 by Claude Shannon in the field of Information Theory. Entropy is the measure of randomness as it represents the uncertainty of the output generated by a data source.

Entropy is denoted by *H(X)* and defined as [10]

$$H(X) = -\sum_{x} P(X = x) \cdot \log_2 (P(X = x)), \quad (1)$$

where $P(X = x)$ is the probability of $X$ taking the value $x$ in the sample space $\Omega_X$.

As shown in [10], the maximum entropy distribution over the range is the uniform distribution. The entropy in this range is no greater than $\log_2|\Omega_X|$.

In cryptographic applications, entropy is one of the measures of the quality of an RNG. Therefore, it is expected to be the highest possible as it translates into more efforts needed by an attacker to guess the output.

*2.3. Cellular Automata (CA).* Stanislaw Ulam took an interest in self-replicating automata concepts in the 1940s and initiated the studies on cellular automata. Later on, upon suggestions by Stanislaw Ulam, John von Neumann further developed the theory on cellular automata in the 1960s to model self-reproduction in the field of Biology. CAs gain popularity in the 1970s with Martin Gardner and John Conway's Game of Life [11].

A cellular automaton represents a network of cells or finite state automaton. The state of each cell at time $t$ changes according to a local rule and depends on the state of the surrounding cells or neighborhood at time $t-1$.

The interest in cellular automata comes from the fact that simple local calculations, at the level of cells, result in a complex global behavior, at the level of the automaton. Even though uniform CAs (uniformity in update, in dependency to neighbors, in local rules, etc.) are suitable for modeling physical systems, nonuniform CAs are more suited and more efficient to model real life systems.

A CA is formally defined as the tuple $(d, L, S, N, f)$ [11], where

(i) d is the dimension of the cellular space

(ii) L is the d-dimensional cellular space, in which elements are called cells

(iii) S is a finite state, in which elements are called states

(iv) $N = (\overrightarrow{v_1}, \overrightarrow{v_2}, \ldots, \overrightarrow{v_N})$ is the neighborhood vector

(v) $f$ is the local rule of the automaton that dictates the change of state of each cell from time $t$ to time $t+1$

If $c$ is considered to be the current configuration of the automaton, then $\Phi(c)$, $\Phi^2(c)$, ... represent its next configurations. $\Phi$ is called the global function.

Changing $d$, $L$, $S$, $N$, and $f$ leads to different types of CAs.

A CA for which all the cells are updated by the same rule $f$ is called uniform. Otherwise, the CA is called hybrid or nonuniform. A CA for which the boundary cells are assigned the state 0 is called a null boundary CA. A periodic boundary CA has the extreme cells adjacent to each other.

Elementary Cellular Automata or ECAs were introduced in the 1980s by Wolfram. These CAs are one-dimensional, have two possible states (0 or 1), and are characterized by a three-neighborhood dependency. Extensive research related to CAs has been conducted in the field of Cryptography,

especially in producing random numbers using CAs. Wolfram introduced this association between Pseudorandom Number Generators and CAs in 1985 [2].

In the case of ECAs, the neighborhood of the $i$th cell is given by $N(i) = \{s_{i-1}, s_i, s_{i+1}\}$ and the local rule $f$ for that cell is defined as $s_i^{t+1} = f_i(s_{i-1}^t, s_i^t, s_{i+1}^t)$, $s_i^t$ being the current state and $s_i^{t+1}$ the next state of the cell. Since ECAs assume two possible states and have a three-neighborhood dependency, $2^{2^3} = 256$ rules are possible. Each of these rules can be represented as a Boolean function or as a truth table, the decimal value of the truth table being the rule number. If the Boolean expression of the rule contains only the XOR ($\oplus$) operator, then the rule is linear. Otherwise, if the Boolean expression contains also AND($\cdot$)/OR($+$) operators, then the rule is nonlinear. Table 1 shows an example of a linear and a nonlinear rule.

## 3. Related Work

Wolfram was the first to use CAs as a source of randomness [2]. To generate pseudorandom numbers sequences (PNSs), he used a one-dimensional cellular automaton with $r = 1$ and rule 30. In [3], Hortensius et al. proposed a PRNG based on cellular automata for built-in self-test. Both uniform (rule 30) and nonuniform (rules 30 and 45, and rules 90 and 150) cellular automata were evaluated in the article. Nandi et al. [4] also worked on nonuniform cellular automata and proposed 5 relevant rulesets (rulesets can be found in Appendix B). Tomassini et al. [5] extended the research on nonuniform, one-dimensional CAs using four rules (90, 105, 150, and 165) selected using cellular programming, an evolutionary technique. Guan et al. introduced controllable CA [6] and self-programmable CA [7]. By using control signals and hybrid cell configurations, they managed to increase the randomness. Seredynski et al. [8] worked also on nonuniform, one-dimensional CAs with rules of radius 1 and 2. They used cellular programming to discover the relevant rules among 47 rules. The most relevant ruleset chosen for radius 1 in [8] was (86, 90, 101, 105, 153, 165). Recently, Bhattacharjee et al. [12] explored 3-state one-dimensional CA. Each of these constructions either has been attacked or presents some flaws. This is still a young field of research, and improvements are still to be made. The configuration of the cellular automaton (dimension, neighborhood, rules selection, etc.) can be seen as an optimization problem. As such, the configuration chosen must be a compromise of cryptographic properties resulting in some flaws and gaps. In this article, the authors chose to work with a hybrid one-dimensional cellular automaton combined with other cryptographic primitives to maximize the strength and the randomness quality of the sequences generated. Table 2 summarizes the previously developed CA-based PRNGs.

### 3.1. CFA Design Principles and Description

*3.1.1. Design Principles.* The goal behind the general *CFA* scheme was to develop a flexible PRNG. In other words, it can be adapted according to the design constraints and can be easily integrated in a broader system. This implies that its components are independent. It also implies that it can rely on existing cryptographic primitives or use new constructions.

*CFA* was also designed with portability and resistance to attacks in mind. The system was conceived to be resistant to most of the known attacks against PRNGs. However, the authors also tried to make it comply with the following constraints:

(i) Flexibility
(ii) Efficiency in execution
(iii) Ease of use
(iv) Simplicity in design

The general building blocks of *CFA* are as follows:

(i) A seed file containing high entropy sequences
(ii) A reseed trigger mechanism that initiates the reseed mechanism
(iii) A reseed mechanism that updates the seed file
(iv) An output generation mechanism that comprises the cryptographic primitives and other functions and generates the output

Figure 1 details the general building blocks and mechanisms by which an input is generated using *CFA*.

In the rest of this section, each of these components is detailed.

*3.1.2. Seed File.* High entropy sequences are stored in the seed file before being used in the output generation mechanism. The origin of these sequences depends on the application and the designer of the system. They are appended to the seed file, stored in flash memory or hard disk, and to be used subsequently by the reseed mechanism to be fed to the output generation mechanism.

*3.1.3. Reseed Trigger Mechanism.* The reseed trigger mechanism controls when a reseeding is necessary based on some parameter threshold. This parameter could be, for example, the period of the cellular automaton. In addition to this periodic reseed, an upon request reseed mechanism can be implemented for when the state of the PRNG is compromised by some attack.

*3.1.4. Reseed Mechanism.* During the reseed mechanism, the seed file is updated. When the reseed mechanism is triggered, a new random sequence of fixed length is generated and prepended to the seed file. When all the sequences of the seed file are used, its content is cleared, and new sequences are generated.

*3.1.5. Output Generation Mechanism.* The output generation mechanism is the core of the system. It uses cryptographic primitives and cellular automata configurations to

Table 1: An example of a linear and nonlinear ECA rule.

| Rule | Linear? | Algebraic normal form | Truth table | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 110 | No | $s_i^{t+1} = s_i^t \oplus s_{i+1}^t \oplus (s_i^t \cdot s_{i+1}^t) \oplus (s_{i-1}^t \cdot s_i^t \cdot s_{i+1}^t)$ | 111<br>0 | 110<br>1 | 101<br>1 | 100<br>0 | 011<br>1 | 010<br>1 | 001<br>1 | 000<br>0 |
| 150 | Yes | $s_i^{t+1} = s_{i-1}^t \oplus s_i^t \oplus s_{i+1}^t$ | 111<br>1 | 110<br>0 | 101<br>0 | 100<br>1 | 011<br>0 | 010<br>1 | 001<br>1 | 000<br>0 |

Table 2: CA-based PRNGs.

| CA- based PRNGs | Cellular automaton | Ruleset |
|---|---|---|
| Wolfram [2] | 2-state one-dimensional CA with $r = 1$ (uniform) | Rule 30 |
| Hortensius et al. [3] | 2-state one-dimensional CA with $r = 1$ (uniform)<br>2-state one-dimensional CA with $r = 1$ (nonuniform) | Rule 30<br>30, 45, 90, 150 |
| Nandi et al. [4] | 2-state one-dimensional CA with $r = 1$ (nonuniform) | 153, 153, 153, 153, 51, 51, 51, 51<br>195, 195, 195, 195, 51, 51, 51, 51<br>51, 51, 153, 153, 153, 153, 51, 51<br>51, 51, 195, 195, 195, 195, 51, 51<br>51, 153, 153, 153, 153, 153, 153, 51 |
| Tomassini et al. [5] | 2-state one-dimensional CA with $r = 1$ (nonuniform) | 90, 105, 150, 165 |
| Guan et al [6, 7] | 2-state one-dimensional controllable CA and self-programmable CA | 90, 165, 150, 105 |
| Seredynski et al. [8] | 2-state one-dimensional CA with $r = 1$ (nonuniform) | 86, 90, 101, 105, 153, 165 |
| Bhattacharjee et al. [12] | 3-state one-dimensional CA with $r = 1$ | 120021120021021120021021210 |

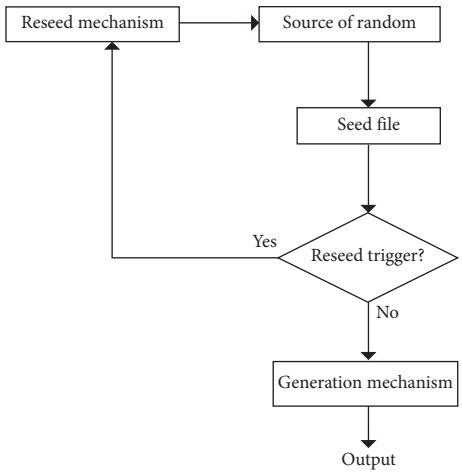



Figure 1: *CFA* block diagram.

generate a random sequence with entropy as high as possible.

*3.2. Description.* In this section, a more detailed description of the general scheme's (*CFA*) generation mechanism and a specific implementation are presented.

*3.2.1. CFA Outline.* *CFA* output generation mechanism comprises three major building blocks:

(i) A hash function, $h(x)$, that has the properties of preimage resistance, second preimage resistance, and collision resistance

(ii) A CA evolution function, $Evol(x)$

(iii) A block cipher, $E(x)$, with good statistical properties and good resistance to attacks

*CFA* has a strength in bits of $\min(m, k)$, where $m$ is the size in bits of the hash function digest, and $k$ is the size in bits of the key of the block cipher. Figure 2 depicts the output generation mechanism of *CFA* comprising $h$, *Evol* and $E$.

*3.2.2. A Specific Implementation: CFA-256.* In *CFA*-256, the building blocks used are as follows:

(i) A seed generated by means of the Bouncy Castle Java library

(ii) The SHA-3-256 hash function

(iii) A one-dimensional, nonuniform two-state CA with $r = 1$

(iv) The AES-256 block cipher in counter mode

*3.2.3. Step 0: Seed File Generation.* The *CFA*-256 algorithm starts with the generation of a seed using the ThreadedSeedGenerator class found in the Bouncy Castle Java library. The size of the seed generated is 1024 bits. Each sequence generated is then stored in the seed file. A selected sequence from the seed file is then fed to the SHA-3 hash function.

*3.2.4. Step 1: SHA-3 Processing.* SHA-3 relies on a sponge construction rather than a Merkle–Damgård construction like SHA-1 and SHA-2. The message is divided into blocks, and padding is applied if necessary (preprocessing phase).

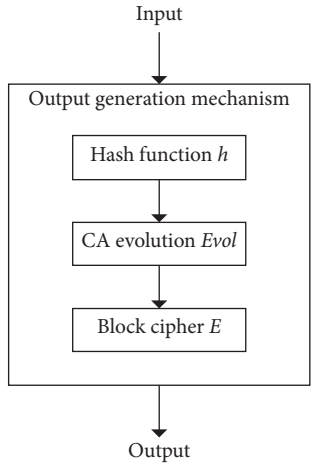



Figure 2: Output Generation Mechanism block diagram.

The sponge construction starts with an absorbing phase followed by a squeezing phase. During the absorbing phase (or input phase), the input blocks $x_i$ are processed, while, during the squeezing phase (or output phase), the output $h(M)$ is computed. For both of these phases, the same $f$ function is used. $f$ is a fixed permutation function consisting of 24 rounds of 5 reversible operations ($\theta, \rho, \pi, \chi$ and $\iota$ applied in this order).

In Figure 3, $r$ represents the size of the input blocks, called the bitrate. $c$ represents the capacity, which is closely related to the security level of the construction and equal to two times the digest size. The sum of these two parameters represents the width of the state or $b$. In SHA-3, $b$ is equal to 1600 bits. Using SHA-3, hash values of lengths 224, 256, 384, or 512 bits can be obtained from messages of any size. For *CFA*-256, the parameters used are $r = 1088$ bits, $c = 512$ bits, and 256 bits for the size of the digest (Table 3, [13]). Table 3 summarizes SHA-3 parameters for different digest sizes.

A 256-bit digest is obtained from the SHA-3 function and is then evolved using a one-dimensional elementary CA.

*3.2.5. Step 2: CA Evolution.* The CA used in this case is a hybrid CA. The local rule $f$ consists of the ruleset [30, 90, 150, 30, 110, 30, 90, 150] applied alternately on the cells. In order to choose those rules, the guideline provided in [1] was followed. The rule selection procedure is detailed in the Appendix. The CA is evolved using this local rule 128 times. The obtained 256 cells are then supplied to the AES encryption algorithm.

*3.2.6. Step 3: AES Encryption.* This step encrypts the output of the second one in counter mode using a random secret key. Figure 4 shows the general AES encryption process. The plaintext is the result of the CA evolutions, and the ciphertext is the random sequence generated by the whole *CFA*-256 algorithm. The secret key involved in the encryption is generated using ThreadedSeedGenerator.

*3.2.7. CFA-256 Algorithm.* Algorithm 1 shows the process by which the seed file is filled with high entropy sequences.

Algorithm 2 shows the generation mechanism of *CFA*-256.

## 4. Security Analysis and Results

*4.1. Security Analysis.* Here, it is assumed that the security of cryptographic primitives translates to the PRNG [14].

According to [15], the following points represent some of the reasons that lead to a compromised PRNG:

(i) Entropy overestimation and guessable starting points
(ii) Chosen-Input Attacks
(iii) Side-Channel Attacks
(iv) Direct Cryptanalytic attacks

*4.1.1. Entropy Overestimation and Guessable Starting Points.* In order to avoid this situation, a seed file has been used, and a seed/reseed mechanism has been implemented [16]. The seed file contains high entropy sequences that are used as input to the hash function. The seed/reseed mechanism is responsible for generating a new state of the PRNG from time to time or upon request. Thus, guessing the starting configuration, even in the case of entropy overestimation, is costly. In [4, 5, 8], no seed generation mechanism, reseed mechanism, or seed is mentioned.

*4.1.2. Chosen-Input Attacks.* To resist these attacks, a cryptographic hash function is used in the generation mechanism. In [4, 5, 8], no strategy or mechanism is used to avoid or lessen the risk of chosen-input attacks.

*4.1.3. Side-Channel Attacks.* A hybrid CA, with a carefully chosen ruleset combining linear and nonlinear rules with good cryptographic properties, has been used to protect against this kind of attacks. As shown in the next section and Appendix B, the cryptographic properties of the ruleset used in CFA-256 are better than the ones presented in [4, 5, 8].

*4.1.4. Direct Cryptanalytic Attacks.* To prevent this kind of attacks, a secure block cipher along with a hash function has been used. It should be mentioned that attacks related to a state compromise are not considered for the design of a PRNG as these attacks can be prevented by the environment. Measures to prevent them should be handled at the system level [16]. In [4, 5, 8], no strategy or mechanism is used to avoid or lessen the risk of chosen-input attacks.

## 5. Results

*5.1. Cryptographic Properties of CA.* In this section, the cryptographic properties (algebraic degree, the nonlinearity, the correlation immunity, etc.) of the hybrid ruleset selected (see Appendix) are presented. Here, $x_i$ correspond to the cellular automaton cells. Tables 4 to 8 show some of the

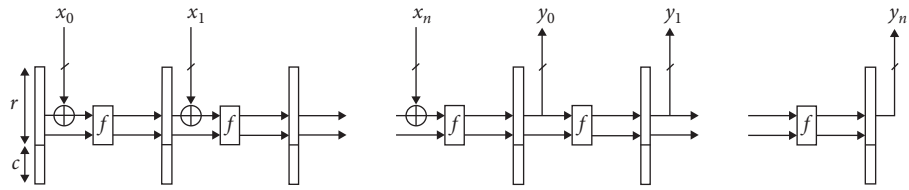



Figure 3: Sponge construction of SHA-3 [9].

Table 3: SHA-3 parameters [13].

| Digest Size | 224 | 256 | 384 | 512 |
|---|---|---|---|---|
| Message size | No maximum | No maximum | No maximum | No maximum |
| Block size | 1152 | 1088 | 832 | 576 |
| Word size | 64 | 64 | 64 | 64 |
| Number of rounds | 24 | 24 | 24 | 24 |
| Capacity c | 448 | 512 | 768 | 1024 |
| Collision resistance | $2^{112}$ | $2^{128}$ | $2^{192}$ | $2^{256}$ |
| Second preimage resistance | $2^{224}$ | $2^{256}$ | $2^{384}$ | $2^{512}$ |

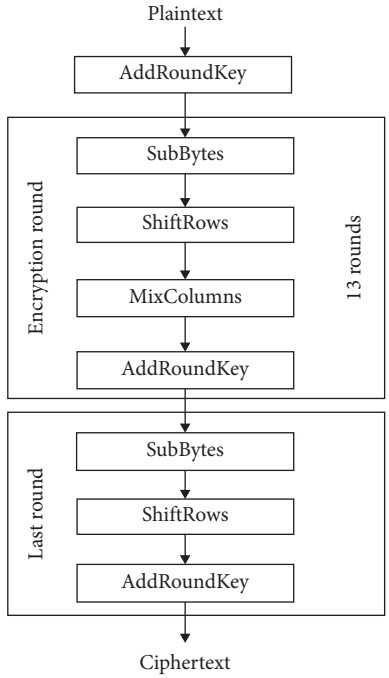



Figure 4: AES-256 encryption [9].

cryptographic properties of the cellular automaton. Here, seven cells and three iterations were used.

From these tables, it can be noted that the correlation immunity, the resiliency, and the balancedness decrease with the iterations. However, the nonlinearity and the algebraic degree increase with the number of clock cycles. Therefore, the ruleset selected appears to be a good compromise for the cryptographic properties (Appendix A displays the selection process). Moreover, if the design proposed in this paper is compared to other known CA-based PRNGs on the basis of cryptographic properties and the CAs evolutions (results are presented in Appendix B and Appendix C), it can be considered as better suited for cryptographic use.

*5.1.1. Avalanche Effect.* One interesting feature sought by any cryptographic primitive is the avalanche effect [9]. This property was first introduced by Feistel [17] and can be expressed as follows: a small change in the input (plaintext or key) results in a large change in the ciphertext [9]. Mathematically, it can be formulated as

$$\forall I, I' | H(I, I') = 1, \text{avalanche}(I') = \frac{H(F(I), F(I'))}{\text{length}(F(I))} \times 100, \tag{2}$$

where $I$ and $I'$ are two inputs that differ in one bit, and $F$ is a function applied to these inputs. In this article, $F$ corresponds to *CFA-256*.

In order to evaluate the avalanche effect of *CFA-256*, the following steps were applied:

(i) Generate 100 high entropy seeds

(ii) For each seed, generate outputs by changing one bit each time

(iii) For each seed, calculate the hamming distances

(iv) For each bit $i$ ($1 \leq i \leq 1024$), calculate the average of the hamming distances

Figure 5 shows the results obtained.

The maximum change is 54.99%, and the minimum is 45.01%. The average of all changes is equal to 50.02%. Therefore, changing a bit in the input yields about 50% changes in the output. This shows that *CFA*-256 displays a good diffusion property.

*5.1.2. NIST Statistical Test Suite.* The NIST Statistical Test Suite (STS) is a set of tests developed by the National Institute of Standards and Technology. The goal behind this test suite is to check the random behavior and the statistical properties of PRNGs and TRNGs algorithms by deriving $p$-values from sequences generated by these algorithms. These $p$-values represent the probability that a sequence was generated by means of a TRNG. A $p$-value measures the

```
Input: number_of_desired_sequences
Output: seed_file
Begin
  If seed_file exists
    Override seed_file
  Else
    Create seed_file
  end if
  Let n ← 0
  While n < number_of_desired_sequences do
    seq[n] ← ThreadedSeedGenerator(128)
    While H(seq[n]) ≤ 0.9 do
      seq[n] ← ThreadedSeedGenerator(128)
    end while
    seed[n] ← seq[n]
    n ← n+1
  end while
  seed_file ← seed
return seed_file
End
```

Algorithm 1: Seed file creation/filling.

```
Input: seed
Output: Random bits (256 bits)
  Begin
  Let j ← (-sizeOf(seed)-2) mod 256
  Let ruleSet ← {30, 90, 150, 30, 110, 30, 90, 150}
  Let key_AES ← ThreadedSeedGenerator(256)
  paddedSeed ← seed || 1||0^j|| 1
  s ← SHA_3(paddedSeed, 256)
  s[0] ← s[256]
  s[257] ← s [1].
  For it from 1 to 128
    For i from 1 to 256
    k ← i–1 modulo sizeOf(ruleSet)
      If ((k = = 0) || (k = = 3) || (k = 5)) then//30
        evolution[i] ← s[i–1] ⊕ (s[i]+s[i+1])
      Else if ((k = = 1) || (k = = 6)) then
        evolution[i] ← s[i–1] ⊕ s[i+1]
      Else if ((k = = 2) || (k = = 7)) then//150
        evolution[i] ← s[i–1] ⊕ s[i] ⊕ s[i+1]
      Else//110
        evolution[i] ← s[i–1] ⊕ s[i+1] ⊕ s[i–1]. s[i+1] ⊕ s[i–1]. s[i]. s[i+1]
      End if
    End for
  End for
  generated_bits ← AES_256 (evolution, key_AES, CTR)
End
```

Algorithm 2: Pseudocode *CFA*-256.

difference between a given sequence and a random sequence. The algorithm passes or fails the tests depending on a significance level, set to 0.01 for the NIST STS. A more detailed description of this test suite is provided in the NIST special publication 800-22 [18].

A sequence of 10 M bits was generated using *CFA*-256 and was used as an input to the test suite. Table 9 shows the results of the tests. *CFA*-256 passes all the applicable tests. Some tests are not applicable due to the fact that the *CFA*-256 output size (256 bits) is too small for the

Table 4: Algebraic degree.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 1 | 2 | 3 | 2 | 1 | 1 |
| 2 | 3 | 2 | 2 | 4 | 4 | 3 | 2 | 2 |
| 3 | 4 | 3 | 4 | 5 | 5 | 4 | 3 | 3 |

Table 5: Nonlinearity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 0 | 2 | 1 | 2 | 0 | 0 |
| 2 | 8 | 8 | 8 | 6 | 10 | 8 | 8 | 8 |
| 3 | 32 | 48 | 52 | 36 | 40 | 48 | 48 | 48 |

Table 6: Correlation immunity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 2 | 0 | 0 | 0 | 1 | 2 |
| 2 | 0 | 2 | 2 | 0 | 0 | 1 | 2 | 2 |
| 3 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 1 |

Table 7: Resiliency.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 2 | 0 | −1 | 0 | 1 | 2 |
| 2 | 0 | 2 | 2 | 0 | −1 | −1 | 2 | 2 |
| 3 | 0 | 0 | 0 | 0 | −1 | −1 | −1 | 1 |

Table 8: Balancedness.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | ✓ | ✓ | ✓ | ✓ | × | ✓ | ✓ | ✓ |
| 2 | ✓ | ✓ | ✓ | ✓ | × | × | ✓ | ✓ |
| 3 | ✓ | ✓ | ✓ | ✓ | × | × | × | ✓ |

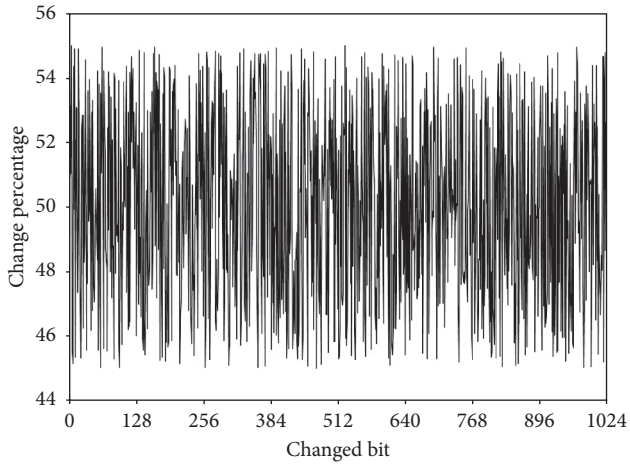



Figure 5: Avalanche effect.

requirements of those tests (1 000 000 bits are required for the nonoverlapping template matching test for example).

*5.1.3. DIEHARDER Battery of Tests.* DIEHARDER is another well-known PRNGs and TRNGs test suite. It is a robust battery of tests developed by Robert G. Brown at Duke University [19]. It was designed to thoroughly test all kinds of random number generators in a systematic way. The goal is to include over time all known statistical tests and develop a universal test suite for PRNGs and TRNGs. As of the latest version available, DIEHARDER includes 17 tests from Georges Marsaglia's DIEHARD test suite, Marsaglia and Tsang GCD test, 3 tests from the NIST STS, 5 tests developed by Brown, and 5 tests developed by David Bauer. The significance level for DIEHARDER is set to 0.005, and smaller $p$-values mean better test results.

Table 9 summarizes the results of the DIEHARDER tests for *CFA*-256 and other known PRNGs. Results in bold represent failed tests. Table 10 shows that *CFA*-256 passes all the tests included in the DIEHARDER test suite. Furthermore, it is the only design that passes all the tests. Therefore, it can be considered as a better alternative to the previously proposed designs.

Table 9: NIST Statistical Test Suite results.

| Test name | $p$ value | Interpretation |
|---|---|---|
| The frequency (monobit) test | 0.5884 | Pass |
| Frequency test within a block | 0.4362 | Pass |
| The runs test | 0.5381 | Pass |
| Tests for the longest-run-of-ones in a block | 0.5448 | Pass |
| The binary matrix rank test | — | Not applicable |
| The discrete Fourier transform (spectral) test | 0.4079 | Pass |
| The nonoverlapping template matching test | — | Not applicable |
| The overlapping template matching test | — | Not applicable |
| Maurer's "universal statistical" test | 0.3946 | Pass |
| The linear complexity test | 0.7515 | Pass |
| The Serial $p$-value 1 test | 0.7857 | Pass |
| The Serial $p$-value 2 test | 0.7030 | Pass |
| The approximate entropy test | 0.7335 | Pass |
| The cumulative sums (cusums) forward test | 0.6292 | Pass |
| The cumulative sums (cusums) reverse test | 0.7458 | Pass |
| The random excursions test | — | Not applicable |
| The random excursions variant test | — | Not applicable |

Table 10: DIEHARDER test suite results.

| Test | CFA | [7] R1 | [7] R2 | [7] R3 | [7] R4 | [7] R5 | [8] | [11] |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.6623 | 0.9802 | 0.8421 | 0.4237 | 0.3459 | 0.9768 | 0.4378 | 0.2232 |
| 2 | 0.5403 | 0.4148 | 0.4755 | 0.2383 | 0.5784 | 0.0202 | 0.2292 | 0.6905 |
| 3 | 0.6715 | 0.8507 | 0.7957 | 0.9124 | 0.4491 | 0.1362 | 0.2419 | 0.3264 |
| 4 | 0.9007 | 0.4125 | **0.0002** | 0.4293 | 0.9708 | 0.5195 | 0.8282 | 0.1726 |
| 5 | 0.2464 | 0.9366 | 0.3630 | 0.6942 | 0.1792 | 0.5380 | 0.8924 | 0.6778 |
| 6 | 0.8277 | 0.0698 | 0.5419 | 0.3899 | 0.6108 | 0.1515 | 0.9086 | **0.0001** |
| 7 | 0.0854 | 0.8900 | 0.4643 | 0.6756 | 0.2325 | 0.1522 | 0.8017 | 0.5926 |
| 8 | 0.4641 | 0.7005 | 0.1575 | 0.0262 | 0.2483 | 0.1222 | 0.5061 | 0.5227 |
| 9 | 0.4858 | 0.3388 | **0.0116** | 0.7640 | 0.6682 | 0.0060 | 0.2537 | 0.9393 |
| 10 | 0.4470 | **0.9960** | 0.9675 | 0.6349 | 0.1520 | 0.3032 | 0.6834 | 0.1153 |
| 11 | 0.2059 | 0.7304 | 0.6101 | 0.9838 | 0.3066 | 0.9427 | 0.9505 | 0.2887 |
| 12 | 0.6967 | 0.7601 | 0.4023 | 0.3048 | 0.3965 | 0.1029 | 0.9210 | 0.3030 |
| 13 | 0.3451 | 0.8119 | 0.9807 | 0.1776 | 0.8243 | 0.5984 | 0.8955 | 0.7754 |
| 14 | 0.5753 | 0.8465 | 0.6718 | 0.7075 | 0.7877 | 0.3422 | 0.8840 | 0.0688 |
| 15 | 0.5935 | 0.0481 | 0.5318 | 0.5789 | 0.2052 | 0.4397 | 0.0267 | 0.0308 |
| 16 | 0.2164 | 0.9461 | 0.9868 | 0.4022 | 0.1908 | 0.0285 | 0.6756 | 0.2968 |
| 17 | 0.3887 | 0.7592 | 0.3333 | 0.9596 | 0.6374 | 0.0793 | 0.7020 | 0.8279 |
| 18 | 0.4494 | 0.4690 | **0.9998** | **0.9968** | 0.7307 | 0.9139 | **0.9982** | 0.4598 |
| 19 | 0.8838 | 0.9109 | 0.9752 | 0.1365 | 0.2587 | 0.7308 | 0.9376 | 0.4891 |
| 20 | 0.5977 | 0.1979 | 0.5449 | 0.5154 | 0.6203 | **0.9966** | 0.5226 | 0.0957 |
| 21 | 0.4952 | 0.1462 | **0.9974** | **0.9986** | 0.2273 | 0.1311 | 0.9534 | 0.6477 |
| 22 | 0.4434 | 0.3337 | 0.7738 | **0.9957** | 0.7480 | 0.6602 | 0.7407 | 0.2266 |
| 23 | 0.4228 | 0.0475 | 0.8172 | 0.1231 | 0.9650 | 0.6983 | 0.1185 | 0.0505 |
| 24 | 0.6120 | 0.4065 | 0.3640 | 0.1977 | **0.0096** | 0.5224 | **0.9992** | **0.9983** |
| 25 | 0.5594 | 0.8959 | **0.9992** | **0.9960** | 0.9813 | 0.2458 | 0.9397 | **0.9961** |
| 26 | 0.6816 | 0.3811 | 0.3171 | 0.6940 | 0.0900 | 0.3590 | 0.5957 | 0.6173 |
| 27 | 0.4042 | 0.2968 | 0.3197 | 0.6481 | 0.9695 | 0.6180 | 0.6694 | 0.3752 |
| 28 | 0.9476 | 0.7612 | 0.1361 | 0.8437 | 0.9904 | 06670 | 0.8914 | 0.0999 |
| 29 | 0.7678 | 0.2497 | 0.1738 | 0.1829 | 0.2446 | 0.7298 | **0.9967** | 0.0461 |
| 30 | 0.7613 | 0.9869 | 0.1332 | **0.0001** | 0.1170 | 0.3688 | 0.9254 | 0.193 |
| 31 | 0.2594 | 0.6661 | 0.8481 | 0.8585 | 0.4841 | 0.5552 | 0.8214 | 0.7918 |

## 6. Challenges, Possible Solutions, and Future Directions

The general design proposed in this article is adaptable, as mentioned before. Consequently, changing the building blocks is done according to applications. However, it is challenging in some applications like IoT systems. It is possible to choose either of the predeveloped lightweight systems (hash functions and block ciphers) together with CAs. As an alternative, irreversible and reversible CAs could

be involved instead of hash functions and block ciphers, respectively, which provide the required properties and security.

Future directions could include the use of CFA in banking systems security like [20]. In addition, the benefit of machine learning to find the best combination of linear and nonlinear CA rules using the classifiers is found in [21]. Moreover, an adequate extension of CFA to secure wireless sensor networks [22] will be proposed.

## 7. Conclusions

A cryptographically secure pseudorandom number generator is useful in various applications. A new family of PRNGs is described in this article. Both the general and the specific designs presented rely on three building blocks carefully selected for their cryptographic features. The generator is built upon two strong cryptographic primitives: a hash function and a block cipher. Those primitives ensure that the PRNG is resistant to several types of attacks. In addition, a hybrid cellular automaton is used to achieve better confusion and diffusion properties, as well as to increase the randomness and security criteria. Passing statistical tests (NIST STS and DIEHARDER), exhibiting a good avalanche property and using a cellular automaton with good cryptographic properties, shows that *CFA*-256 can generate high quality random sequences and displays good confusion and diffusion properties.

## Appendix

## A. Cellular Automata

The self-reproducing feature of cellular automata makes them a good candidate for the generation of high quality random sequences [1]. This randomness property is provided by linear rules only. Although maximum period linear rules provide high quality randomness and security against side-channel attacks (power attack, timing attack, etc.), linear cellular automata have been shown to be insecure for cryptographic applications [23]. Therefore, nonlinear rules are needed to achieve a better security against linear cryptanalysis and MS (Meier and Staffelbach) attack [23]. However, nonlinear rules have high correlation [23]. Consequently, for cryptographic applications, hybrid cellular automata with a ruleset made of a combination of maximum period linear rules and nonlinear rules carefully chosen should be used. For the selection of the ruleset used within the specific pseudorandom generator proposed in the paper, the authors followed the guideline provided in [1].

### A.1. Definitions

Before detailing the selection process of the ruleset used, some definitions related to cryptographic properties are described [24].

*A.1.1. Affine Function.* A Boolean function involving only $\oplus$ combinations of its input. $f(x_1, x_2) = x_1 \oplus x_2$ is an example of an affine function.

*A.1.2. Hamming Weight.* Number 1 in a Boolean function's truth table is called the Hamming weight of the function.

*A.1.3. Hamming Distance.* The hamming weight of $f_1 \oplus f_2$ is called the Hamming distance between $f_1$ and $f_2$. For example, the hamming distance between $f_1(x_1, x_2) = x_1 \oplus x_2$ and $f_2(x_1, x_2) = x_1 \cdot x_2$ is 3.

*A.1.4. Algebraic Degree.* The algebraic degree of a Boolean function is the number of variables in the highest order term with nonzero coefficient. For example, the algebraic degree of $f(x_1, x_2) = x_1 \cdot x_2 \oplus x_2$ is 2.

*A.1.5. Nonlinearity.* The minimum of the Hamming distances between a Boolean function $f$ and all affine functions involving its input variables is known as the nonlinearity of the function. For example, the nonlinearity of $f(x_1, x_2) = x_1 \oplus x_2$ is 0.

*A.1.6. Balancedness.* If the Hamming weight of a Boolean function of $n$ variables is $2^{n-1}$, it is called a balanced Boolean function. $f(x_1, x_2) = x_1 \oplus x_2$ is balanced. $f(x_1, x_2) = x_1 \cdot x_2$ is not balanced.

*A.1.7. Correlation Immunity.* A function in $n$ variables is correlation immunity of order $k$, $1 \le k \le n$, if and only if all of the Walsh transforms $\widehat{f(w)} = \sum_{x \in V_n} (-1)^{f(x) \oplus x \cdot w}$, $1 \le \mathrm{wt}(w) \le k$, are equal zero.

A Boolean function $f$ in $n$ variables is correlation immunity of order $k$ if its values are statistically independent of any subset of $k$ input variables.

*A.1.8. Resiliency.* A Boolean function, which is balanced, and correlation immunity of order $k$ are said to be a $k$-resilient function. For example, the resiliency of $f(x_1, x_2) = x_1 \oplus x_2$ is 1, and the resiliency of $f(x_1, x_2) = x_1 \cdot x_2$ is 0.

*A.1.9. Selection Process.* In [1], a selection procedure for selecting a ruleset for constructing a robust hybrid cellular automaton is presented. Table 11 summarizes the different steps of the guideline along with the choices made by the authors for each of these steps. The sets of linear rules and nonlinear rules are taken from [1, 25]. Tables 12 and 13 summarize their algebraic normal form and their cryptographic properties up to the third iteration.

## B. Cryptographic Properties

The following tables summarize the cryptographic properties of some previously proposed CA-based pseudorandom number generators.

Table 11: Selection process for the ruleset.

| Principle | Choice |
|---|---|
| Choose a nonlinear rule with good cryptographic properties for the first cell | Rule 30 selected [30] |
| In order to have a good compromise between the algebraic degree and the correlation immunity, an equal number of linear and nonlinear rules should be selected for the remaining cells | Rules 60, 90, and 110 selected [30, 60, 110, 90] |
| To increase the algebraic degree, two or three nonlinear rules should be selected consecutively | Rule 30 followed by rule 110 [30, 60, 30, 110, 90] |
| To increase the order of correlation immunity, one nonlinear rule should be followed by more than one linear rule | Rule 30 followed by rules 60 and 90 [30, 60, 90, 30, 110, 30, 60, 90] |
| To increase the period of the hybrid CA, rules with larger period should be chosen | Rules 90 and 150 selected [30, 90, 150, 30, 110, 30, 90, 150] |

Input: the set of linear rules $L = \{60, 90, 102, 105, 150, 165, 195\}$ and the set of nonlinear rules $NL = \{22, 30, 37, 41, 43, 45, 91, 110, 120, 135, 180, 210\}$. Output: a ruleset with rules having good cryptographic properties [30, 90, 150, 30, 110, 30, 90, 150].

Table 12: ANF of rules.

| Rule | Linear? | ANF |
|---|---|---|
| 22 | No | $x_{i-1} \cdot x_i \cdot x_{i+1} \oplus x_{i-1} \oplus x_i \oplus x_{i+1}$ |
| 30 | No | $x_i \cdot x_{i+1} \oplus x_{i-1} \oplus x_i \oplus x_{i+1}$ |
| 37 | No | $1 \oplus x_{i-1} \cdot x_i \cdot x_{i+1} \oplus x_{i-1} \oplus x_{i+1}$ |
| 41 | No | $1 \oplus x_{i-1} \cdot x_i \oplus x_{i-1} \cdot x_i \cdot x_{i+1} \oplus x_{i-1}$ |
| 43 | No | $1 \oplus x_{i-1} \cdot x_i \oplus x_{i-1} \cdot x_{i+1} \oplus x_i \cdot x_{i+1} \oplus x_{i-1} \oplus x_i$ |
| 45 | No | $1 \oplus x_i \cdot x_{i+1} \oplus x_{i-1} \oplus x_{i+1}$ |
| 60 | Yes | $x_{i-1} \oplus x_i$ |
| 90 | Yes | $x_{i-1} \oplus x_{i+1}$ |
| 91 | No | $1 \oplus x_{i-1} \cdot x_i \oplus x_{i-1} \cdot x_{i+1} \oplus x_i \cdot x_{i+1} \oplus x_{i-1} \cdot x_i \cdot x_{i+1} \oplus x_i$ |
| 102 | Yes | $x_i \oplus x_{i+1}$ |
| 105 | Yes | $1 \oplus x_{i-1} \oplus x_i \oplus x_{i+1}$ |
| 110 | No | $x_{i-1} \cdot x_i \cdot x_{i+1} \oplus x_i \cdot x_{i+1} \oplus x_i \oplus x_{i+1}$ |
| 120 | No | $x_{i-1} \oplus x_i \cdot x_{i+1}$ |
| 135 | No | $1 \oplus x_i \cdot x_{i+1} \oplus x_{i-1}$ |
| 150 | Yes | $x_{i-1} \oplus x_i \oplus x_{i+1}$ |
| 165 | Yes | $1 \oplus x_{i-1} \oplus x_{i+1}$ |
| 180 | No | $x_{i-1} \oplus x_i \oplus x_i \cdot x_{i+1}$ |
| 195 | Yes | $1 \oplus x_{i-1} \oplus x_i$ |
| 210 | No | $x_{i-1} \oplus x_{i+1} \oplus x_i \cdot x_{i+1}$ |

Table 13: Cryptographic properties of rules.

| Rule | Algebraic degree | Nonlinearity | Balanced | Correlation immunity |
|---|---|---|---|---|
| 22 | 7 | 45 | No | 1 |
| 30 | 5 | 40 | Yes | 1 |
| 37 | 7 | 49 | No | 0 |
| 41 | 7 | 44 | No | 1 |
| 43 | 5 | 44 | Yes | 0 |
| 45 | 4 | 40 | Yes | 1 |
| 60 | 1 | 0 | Yes | 3 |
| 90 | 1 | 0 | Yes | 3 |
| 91 | 7 | 49 | No | 0 |
| 102 | 1 | 0 | Yes | 3 |
| 105 | 1 | 0 | Yes | 4 |
| 110 | 6 | 38 | No | 1 |
| 120 | 5 | 48 | Yes | 0 |
| 135 | 5 | 48 | Yes | 2 |
| 150 | 1 | 0 | Yes | 4 |
| 165 | 1 | 0 | Yes | 3 |
| 180 | 4 | 32 | Yes | 1 |
| 195 | 1 | 0 | Yes | 3 |
| 210 | 4 | 32 | Yes | 0 |

Table 14: Nonlinearity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 0 | 2 | 0 | 0 | 0 | 2 | 0 |
| 2 | 8 | 12 | 8 | 8 | 0 | 0 | 8 | 8 |
| 3 | 32 | 48 | 48 | 32 | 32 | 32 | 32 | 32 |

Table 15: Algebraic degree.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 3 | 2 | 1 | 1 | 3 | 2 |
| 3 | 2 | 3 | 4 | 3 | 2 | 3 | 3 | 3 |

Table 16: Correlation immunity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 2 | 1 | 1 | 0 | 1 |
| 2 | 1 | 0 | 0 | 0 | 1 | 2 | 1 | 0 |
| 3 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

Table 17: Resiliency.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 2 | 1 | 1 | 0 | 1 |
| 2 | 1 | 0 | 0 | 0 | 1 | 2 | 1 | 0 |
| 3 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

Table 18: Balancedness.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 19: Nonlinearity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 20: Algebraic degree.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Seredynski et al. [8] Ruleset Cryptographic Properties. Ruleset: 86, 90, 101, 105, 153, 165.

Tables 14 to 18 show the computed cryptographic properties (nonlinearity, algebraic degree, correlation immunity, resiliency, and balancedness) for the ruleset used in [8].

Tomassini et al. [5] Ruleset Cryptographic Properties. Ruleset: 90, 105, 150, 165.

Table 21: Correlation immunity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 1 |
| 2 | 2 | 3 | 3 | 2 | 2 | 3 | 3 | 2 |
| 3 | 3 | 4 | 2 | 4 | 4 | 2 | 2 | 3 |

Table 22: Resiliency.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 1 |
| 2 | 2 | 3 | 3 | 2 | 2 | 3 | 3 | 2 |
| 3 | 3 | 4 | 2 | 4 | 4 | 2 | 2 | 3 |

Table 23: Balancedness.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 24: Nonlinearity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 25: Algebraic degree.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 26: Correlation immunity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 3 | 3 | 3 | 2 | 0 | 0 | 0 | 0 |

Table 27: Resiliency.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 3 | 3 | 3 | 2 | 0 | 0 | 0 | 0 |

Tables 19 to 23 show the computed cryptographic properties (nonlinearity, algebraic degree, correlation immunity, resiliency, and balancedness) for the ruleset used in [5].

Nandi et al. [4] Ruleset Cryptographic Properties.

Tables 24 to 28 show the computed cryptographic properties (nonlinearity, algebraic degree, correlation

Table 28: Balancedness.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 29: Nonlinearity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 30: Algebraic degree.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 31: Correlation immunity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 1 | 1 | 3 | 3 | 0 | 0 | 0 | 0 |

Table 32: Resiliency.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 1 | 1 | 3 | 3 | 0 | 0 | 0 | 0 |

Table 33: Balancedness.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 34: Nonlinearity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

immunity, resiliency, and balancedness) for the ruleset 1 used in [4].

Ruleset 1: 153,153,153,153,51,51,51,51.

Tables 29 to 33 show the computed cryptographic properties (nonlinearity, algebraic degree, correlation immunity, resiliency, and balancedness) for the ruleset 2 used in [4].

Table 35: Algebraic degree.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 36: Correlation immunity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 1 | 1 | 2 | 0 | 0 |
| 3 | 0 | 0 | 3 | 3 | 2 | 1 | 0 | 0 |

Table 37: Resiliency.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 1 | 1 | 2 | 0 | 0 |
| 3 | 0 | 0 | 3 | 3 | 2 | 1 | 0 | 0 |

Table 38: Balancedness.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 39: Nonlinearity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 40: Algebraic degree.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Ruleset 2: 195, 195, 195, 195, 51, 51, 51, 51.

Tables 34 to 38 show the computed cryptographic properties (nonlinearity, algebraic degree, correlation immunity, resiliency, and balancedness) for the ruleset 3 used in [4].

Ruleset 3: 51, 51, 153, 153, 153, 153, 51, 51.

Tables 39 to 43 show the computed cryptographic properties (nonlinearity, algebraic degree, correlation immunity, resiliency, and balancedness) for the ruleset 4 used in [4].

Ruleset 4: 51, 51, 195, 195, 195, 195, 51, 51.

Tables 44 to 48 show the computed cryptographic properties (nonlinearity, algebraic degree, correlation immunity, resiliency, and balancedness) for the ruleset 5 used in [4].

Ruleset 5: 51, 153, 153, 153, 153, 153, 153, 51.

## C. Cellular Automata Evolutions

Table 49 shows the evolutions of the cellular automaton building block of CFA-256 and other previously proposed CA-based PRNGs. The results shown are for 256 cells evolved for 128 time-steps.

Table 41: Correlation immunity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 2 | 3 | 3 | 0 | 0 |

Table 42: Resiliency.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 2 | 3 | 3 | 0 | 0 |

Table 43: Balancedness.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 44: Nonlinearity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 45: Algebraic degree.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 46: Correlation immunity.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 2 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 0 |
| 3 | 0 | 3 | 3 | 3 | 3 | 2 | 1 | 0 |

Table 47: Resiliency.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 2 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 0 |
| 3 | 0 | 3 | 3 | 3 | 3 | 2 | 1 | 0 |

Space-time diagrams are used as a visual tool to study the behavior of cellular automata and look for repetitive patterns. From Table 49, all the rulesets presented in [4] display repetitive patterns. The other designs do not seem to display repetitive patterns and have a balanced distribution of 0 s and 1 s.

Table 48: Balancedness.

| Iteration | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|---|---|
| 1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 49: Evolutions of different designs.

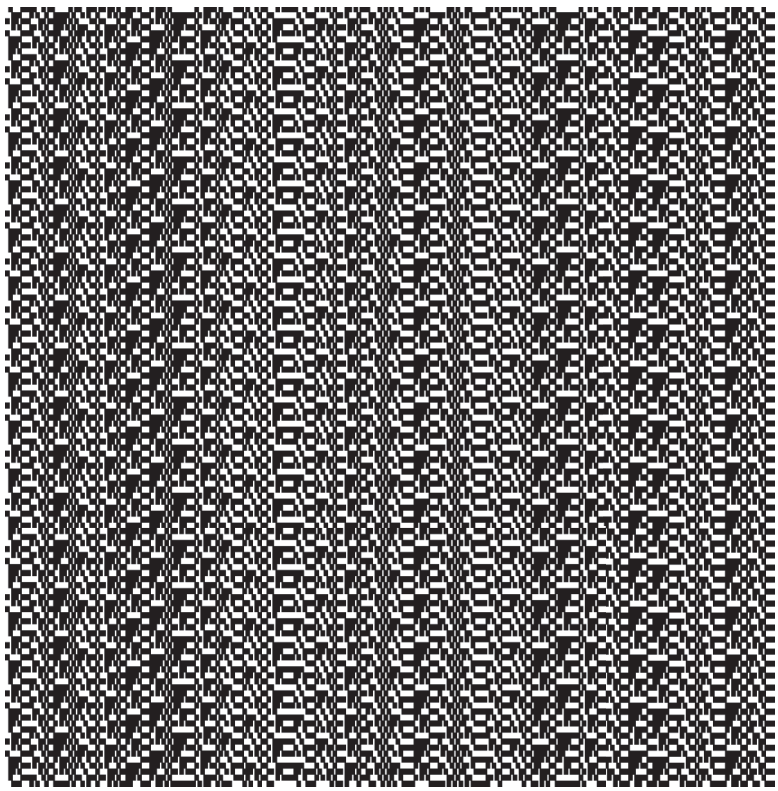
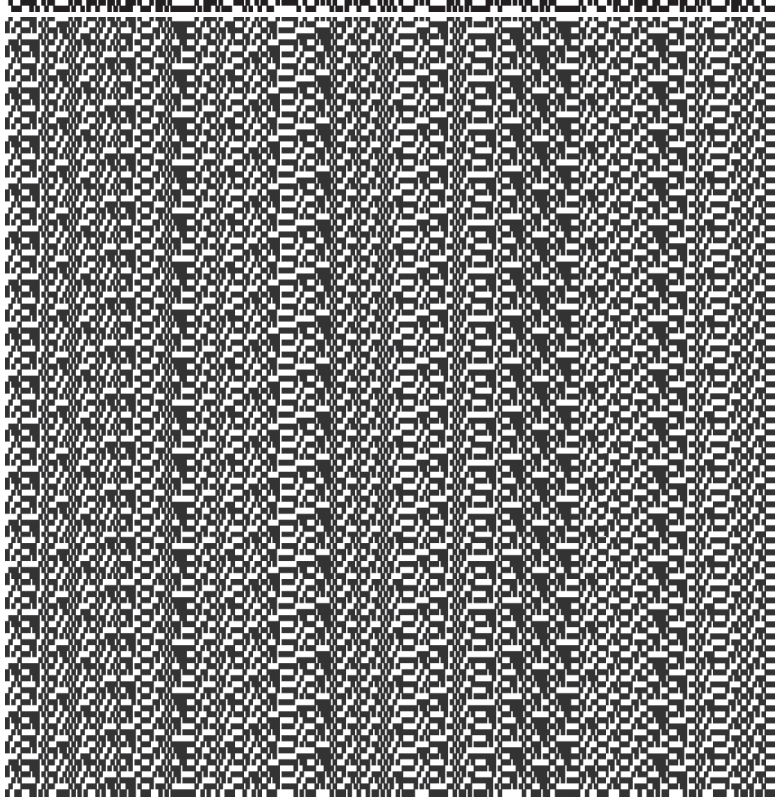
| | |
|---|---|
| Nandi et al. [4] Ruleset 1 |  |
| Nandi et al. [4] ruleset 2 |  |

Table 49: Continued.

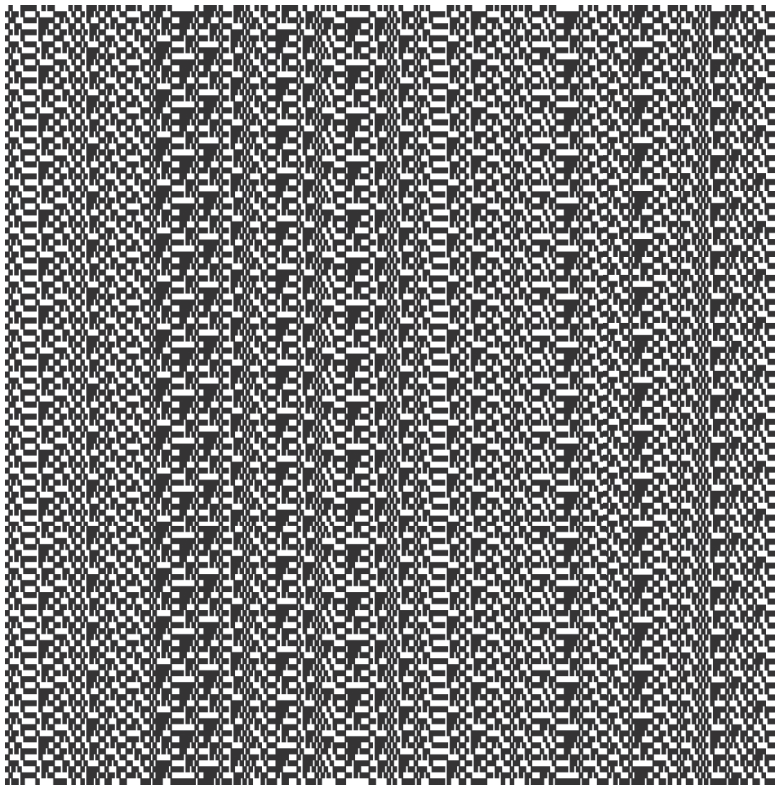
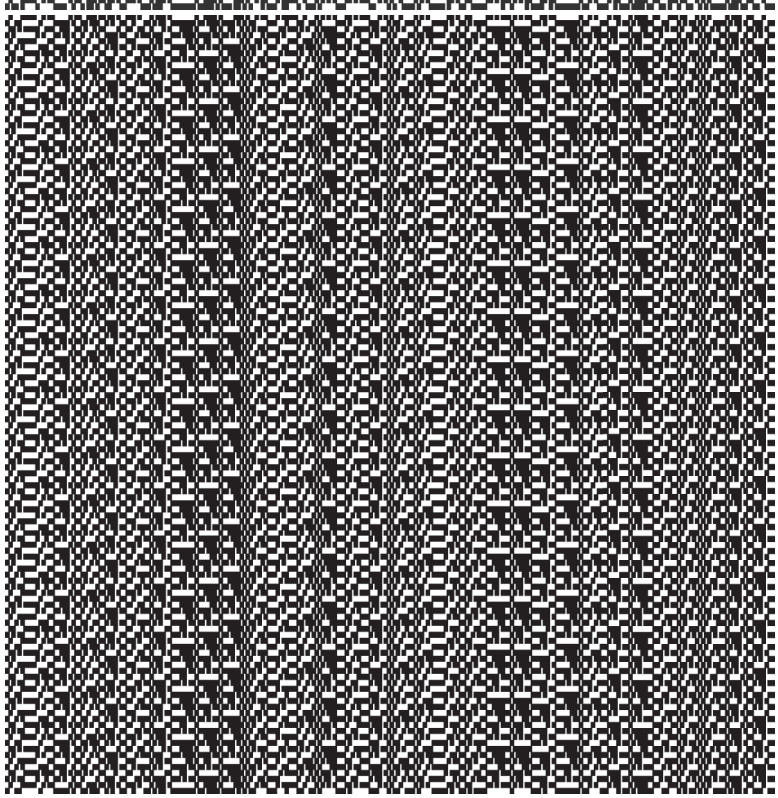
| | |
|---|---|
| Nandi et al. [4] ruleset 3 |  |
| Nandi et al. [4] ruleset 4 |  |

Table 49: Continued.

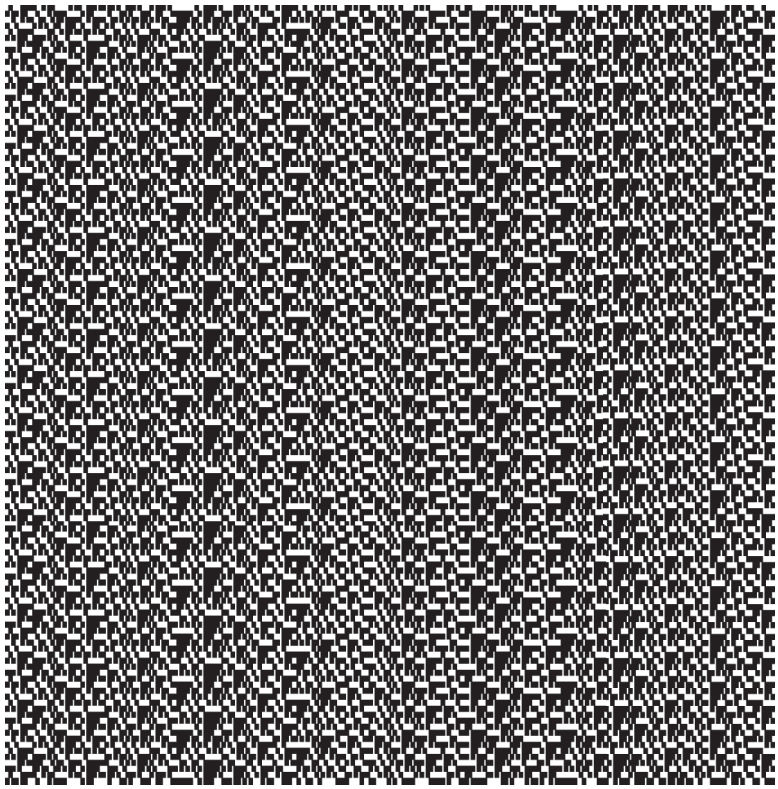
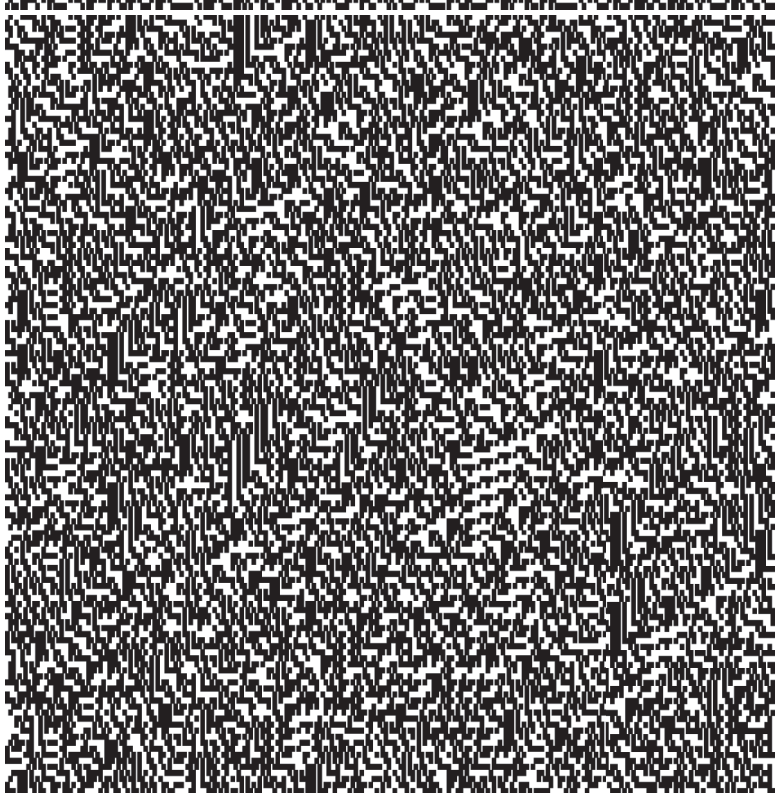
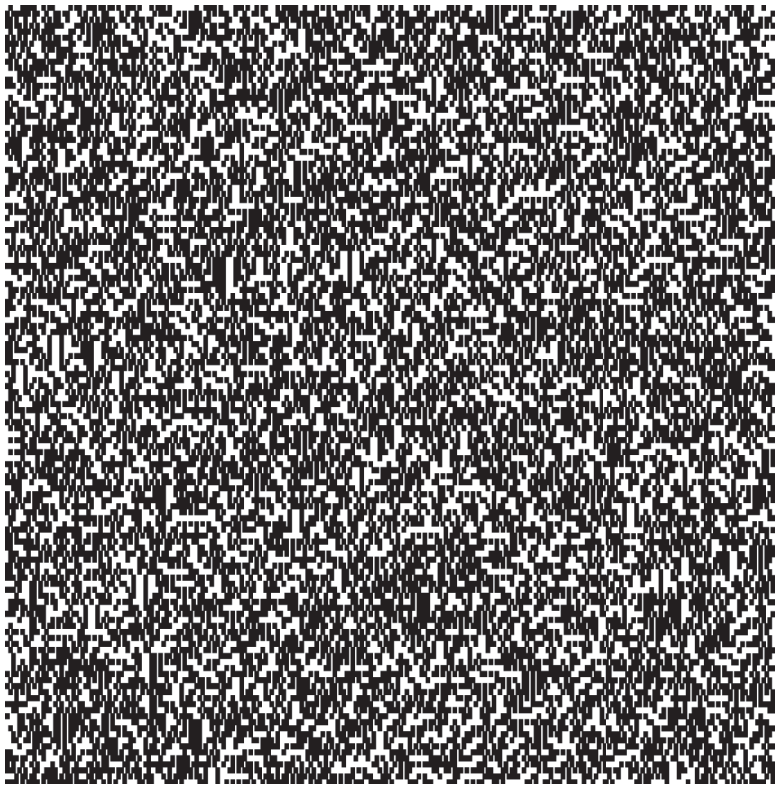
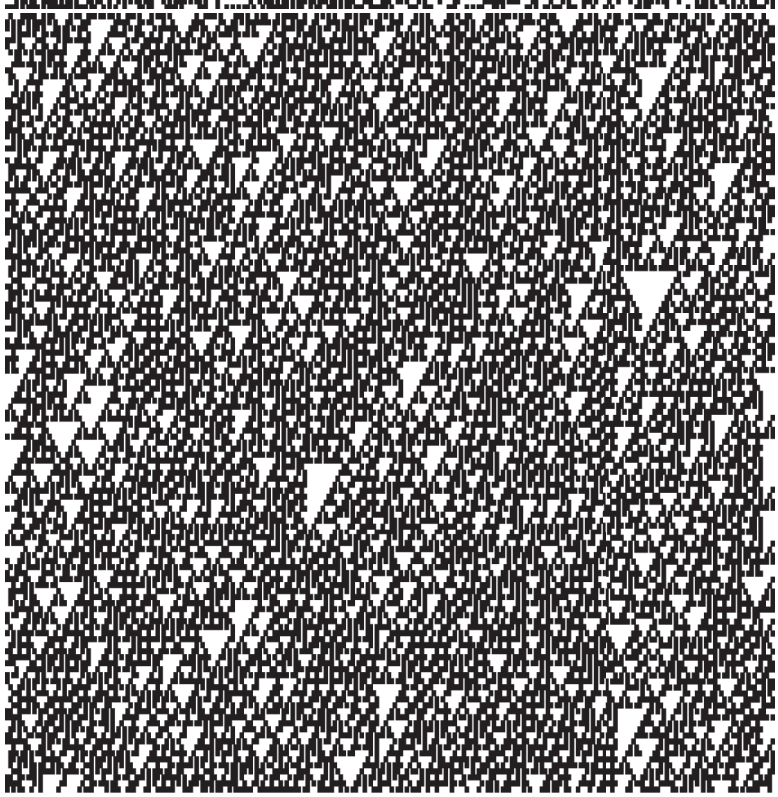
| | |
|---|---|
| Nandi et al. [4] ruleset 5 |  |
| Tomassini et al. [5] |  |

Table 49: Continued.

| | |
|---|---|
| Seredynski et al. [8] |  |
| CFA-256 CA |  |

## Data Availability

The data used to support the findings of this study are included in the article.

## Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

## References


[1] K. Chakraborty and D. R. Chowdhury, "CSHR: selection of cryptographically suitable hybrid cellular automata rule," *Lecture Notes in Computer Science*, vol. 33, pp. 591–600, 2012.

[2] S. Wolfram, "Origins of randomness in physical systems," *Physical Review Letters*, vol. 55, no. 5, pp. 449–452, 1985.

[3] P. D. Hortensius, R. D. Mcleod, W. Pries, D. M. Miller, and H. C. Card, "Cellular automata-based pseudorandom number generators for built-in self-test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 8, pp. 842–859, 1989.

[4] S. Nandi, B. K. Kar, and P. Pal Chaudhuri, "Theory and applications of cellular automata in cryptography," *IEEE Transactions on Computers*, vol. 43, no. 12, pp. 1346–1357, 1994.

[5] M. Tomassini, M. Sipper, M. Zolla, and M. Perrenoud, "Generating high-quality random numbers in parallel by cellular automata," *Future Generation Computer Systems*, vol. 16, no. 2-3, pp. 291–305, 1999.

[6] S. U. Guan and S. Zhang, "An evolutionary approach to the design of controllable cellular automata structure for random number generation," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 1, pp. 23–36, 2003.

[7] S.-U. Guan and S. K. Tan, "Pseudorandom number generation with self-programmable cellular automata," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 7, pp. 1095–1101, 2004.

[8] F. Seredynski, P. Bouvry, and A. Y. Zomaya, "Cellular automata computations and secret key cryptography," *Parallel Computing*, vol. 30, no. 5-6, pp. 753–766, 2004.

[9] W. Stallings, "Cryptography and network security: principles and practice," 2017.

[10] T. M. Cover and J. A. Thomas, "Elements of information theory," 2012.

[11] K. Bhattacharjee, N. Naskar, S. Roy, and S. Das, "A survey of cellular automata: types, dynamics, non-uniformity and applications," *Natural Computing*, vol. 29, 2018.

[12] K. Bhattacharjee, D. Paul, and S. Das, "Pseudo-random number generation using a 3-state cellular automaton," *International Journal of Modern Physics C*, vol. 28, no. 6, Article ID 1750078, 2017.

[13] W. Stallings and L. Brown, "Computer security: principles and practice," 2017.

[14] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Cryptanalytic attacks on pseudorandom number generators," *International Workshop on Fast Software Encryption*, vol. 188, 1998.

[15] J. Kelsey, B. Schneier, and N. Ferguson, "Yarrow-160: notes on the design and analysis of the yarrow cryptographic pseudorandom number generator," *Selected Areas in Cryptography Lecture Notes in Computer Science*, vol. 33, 2000.

[16] P. Gutmann, "Software generation of practically strong random numbers," in *Proceedings of the Usenix Security Symposium*, Austin, TX, USA, August 1998.

[17] H. Feistel, "Cryptography and computer privacy," *Scientific American*, vol. 228, no. 5, pp. 15–23, 1973.

[18] A. Rukhin, J. Sota, J. Nechvatal, M. Smid, E. Barker, and S. Leigh, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," 2000.

[19] R. G. Brown, D. Eddelbuettel, and D. Bauer, "Dieharder: a random number test suite," 2013.

[20] N. Etaher, G. R. S. Weir, and M. Alazab, "From zeus to zitmo: trends in banking malware," *IEEE Trustcom/BigDataSE/ISPA*, vol. 11, pp. 1386–1391, 2015.

[21] R. U. Khan, X. Zhang, R. Kumar, A. Sharif, N. A. Golilarz, and M. Alazab, "An adaptive multi-layer botnet detection technique using machine learning classifiers," *Applied Sciences*, vol. 9, no. 11, p. 2375, 2019.

[22] M. Numan, F. Subhan, W. Z. Khan et al., "A systematic review on clone node detection in static wireless sensor networks," *IEEE Access*, vol. 8, pp. 65450–65461, 2020.

[23] S. Karmakar, D. Mukhopadhyay, and D. R. Chowdhury, "D-monomial tests of nonlinear cellular automata for cryptographic design," *International Conference on Cellular Automata*, vol. 270, 2010.

[24] C. Carlet, "Boolean functions for cryptography and error correcting codes," *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, vol. 35, pp. 257–397, 2010.

[25] G. Maity and J. Bhaumik, "New hybrid CA rule sets for cryptographic design," in *Proceedings of the 2015 Third International Conference on Computer, Communication, Control and Information Technology (C3IT)*, pp. 1–5, Hooghly, India, February 2015.