A Hardware and Secure Pseudorandom Generator for Constrained Devices

Mohammed Bakiri, Christophe Guyeux, Jean-François Couchot, Luigi Marangio, and Stefano Galatolo

***Abstract*—Hardware security for an Internet of Things (IoT) or cyber physical system drives the need for ubiquitous cryptography to different sensing infrastructures in these fields. In particular, generating strong cryptographic keys on such resource-constrained device depends on a lightweight and cryptographically secure random number generator. In this research work, we have introduced a new hardware chaos-based pseudorandom number generator, which is mainly based on the deletion of an Hamilton cycle within the N-cube (or on the vectorial negation), plus one single permutation. We have rigorously proven the chaotic behavior and cryptographically secure property of the whole proposal: the mid-term effects of a slight modification of the seed (proven to be sensitive to the initial conditions) or of the inputted generator cannot be predicted. The proposal has been fully deployed on a FPGA and 65nm ASIC, it runs completely in parallel while consuming as low resources as possible, and achieving: (a) 11.5 Gbps for FPGA and 9.4 Gbps for ASIC random bit throughput, (b) $3.3\mu W$ (LF) to $7.8mW$ (UHF) total power consumption with 5% leakage power, measured at $1.32V$, and (c) able to successfully pass the statistical tests of NIST and TestU01 (BigCrush).**

***Index Terms*—Random number generators, Chaotic circuits, Discrete dynamical systems, Statistical tests, Lightweight Cryptography, Constrained devices, Applied cryptography, FPGA.**

I. INTRODUCTION

SEcurity and cryptography are key elements in Internet of Things constrained devices and these elements are challenging since they face a wider range of limitation [1], [2], including energy (dynamic and static power), latency (delay to complete a process), throughput (rate in bps), and scalability [3]. Hardware security is still in its infant stage, and a lot of technical difficulties associated with IoT need to be overcome [4]. In this context, as most protocols rely on the security of a good random number generator (*e.g.*, key establishment, authentication, etc. [5]), this latter appears as a key element in lightweight security core for IoT devices. However, only a few research works focus on such hardware random number generators, and no standard is currently available.

Such generators are usually divided in two categories: "pseudorandom" number generators (PRNGs), which use deterministic algorithms to produce numbers that look like random (they pass statistical tests with success), and "true" random number generators (TRNGs) that employ a physical source of entropy to produce randomness. Despite the intrinsic quality of TRNGs, most of these techniques are however implemented in a manner that is either slow (*i.e*, in a range of some Kbps to Mbps, to extract noise or jitter from a given component [6]) or costly (*e.g.*, extracting or measuring some noise using oscilloscope or laser [7]). A first alternative solution was *Entropy as Service* [8], which generates keys based on quantum effects at IoT boot, and without any possibility for the cloud service to gain any insight into the user keys [9]. However, many countries or enterprises do not have access to this cloud infrastructure (locally or through the Internet), and local solutions are often desired for securing, monitoring, or to communicate data.

A lightweight cryptographic primitive must satisfy a number of specifications and standards as defined for example in ISO/IEC 14223, 14443 [10], or 18000 [11]. For instance, for physical layer applications, we consider areas (in *Gate Equivalent* GE) lower than 5000 [12] [13], dynamic and static power conception in a range of $5\mu A$ to $15\mu A$ and working in different frequency bands (Low Frequency LF of 100Khz, high frequency HF of 13.56MHz, and Ultra high frequency UHF). In a related work [14], an 8-bit PRNG (i-Beam sensor) has initially been proposed that xored the current value of the time with a key, reaching a throughput of 400$Mbps$. However, this PRNG has revealed to have a short period, as described in [15]. TinyRNG [16] is another PRNG that combines two cipher blocks based on chaining message authentication code (CBC-MAC) and on counter mode (CTR), where the first one aims to extract the transmission of bit errors on the network as randomness sources to reseed the second cipher key. In [17], the authors propose a new PRNG named Warbler, in *Electronic product Code* EPC GEN2 RFID Tag [18], that combines a nonlinear feedback shift register (LFSR) with a nonlinear feedback *Welch-Gong* shift register. However, it has been proven in [19] that this non linear LFSR has various vulnerabilities when used as a stream cipher. Furthermore, it generates only 1 bit each 5 cycles, which drops the throughput to 408$Mbps$ (other PRNGs designed for IoT and RFID tags are reported at the end of this paper, for the sake of comparison [20]).

In this article, our objective is to fill the gap of hardware pseudorandom number generation specifically designed for the IoT. To do so, the Chaotic Iterations based PRNGs (CIPRNGs) class of generators will be presented, which can be summarized as follows. At each iteration, a new input is received from another given extremely fast generator, called the strategy. This input is used with an updating function

M. Bakiri is from Centre de Dveloppement des Technologie Avances, Cit 20 aot 1956 Baba Hassen, Algier, Algeria.

C.Guyeux, J-F. Couchot, and L. Marangio, were with the Femto-ST Institute, UMR 6174 CNRS, Université de Bourgogne Franche-Comté, France.

S. Galatolo from Dipartimento di Matematica, University di Pisa, Italy.

This work is supported by of the EU (Feder) and the Swiss Confederation within the framework of the Interreg France-Switzerland programme.

E-mail: mbakiri@cdta.dz.

Manuscript received XXX; revised XXX.

based on the so-called Generalized Chaotic Iterations (GCI), whose graph is strongly connected. This property can been practically established by removing a balanced Hamiltonian cycle in a N-cube following the method suggested in [21] and [22]. Thanks to an embedded Boolean GCI function and a permutation, this generator named GCIPRNG (which can be considered as a post-treatment on the inputted one), has usually a better statistical profile than its input, while running at a similar speed. The chaotic dependence between the input and the output is then proven, as well as the preservation of the cryptographically secure property.

The remainder of this article is organized as follows. The next section recalls various proposals in the use of chaos for hardware pseudorandom number generation. Their FPGA implementation and statistical test analysis are reported. Section III describes our proposed design for a new chaotic PRNG, targeting a FPGA implementation. Section IV mathematically demonstrates the chaotic properties of this proposal. Then, the cryptographically secure proofs are presented in Section V. In Section VI, the hardware platform used to evaluate all evoked chaotic PRNGs is presented. Statistical comparisons are provided in the same section, using the TestU01 battery of tests [23]. This article ends by an ASIC implementation in 65nm (Section VII), while further comparisons for real-world applications are finaly provided.

II. CHAOTIC RANDOM NUMBER GENERATORS

This section first presents an extended list of PRNGs that are linked to a chaotic behavior in one way or another. It next presents their FPGA implementation to compare them in terms of hardware resources and statistical behavior.

Chaotic Mapping PRNGs. In [24], authors have used fixed point representation to implement the logistic map ($x^{t+1} = rx^t(1-x^t)$, where $0 < x^t < 1$ and r is the *biotic potential*, $3.57 < r < 4.0$) using Matlab DSP System Toolbox software. They generate many designs with different lengths from 16 to 64 bits, where the resources are dependent on the precision (24 to 53 bits). Authors of [25] compare this implementation with another chaotic PRNG based on the Hénon map [26]. Unlike the logistic map, DSP blocks of the FPGA for all multiplications needed to implement the value $a(x^t)^2$ of this map. Two optimized versions of PRNGs based on chaotic logistic map are proposed in [27] too, which aims to pipeline the multiplication while adding some delays into each stage.

In [28], the authors vary the biotic potential r and observe the divergence of random for almost all initial values. Accordingly, they propose a range of the form $[\alpha, 1-\alpha]$, where $\alpha < 0.5$. Another way to select the parameter r is presented in [29] in which the authors propose a couple of two logistic map PRNGs, each having different seeds and parameters. The main idea is to recycle the pseudorandom number generated by the first chaotic map, namely x^{t+1}, as the biotic potential r_2 for the second one (y^{t+1}) when either $3.57 < x^{t+1} < 4$ is satisfied or the sequence output is divergent. Finally, in [30] four different chaotic maps are implemented in FPGA, namely, the so-called *Bernoulli*, *Chebychev*, *Tent*, and *Cubic* maps.

Chaotic based Timing Reseeding (CTR) PRNGs. Authors of [31] address the short period problem due to the quantization error from a nonlinear chaotic map PRNG. Instead of initializing the chaotic PRNG with a new seed, the seed can be selected by masking the current state x^{t+1} at a specific time. This main concept of CTR was first implemented in FPGA [32], in which the critical path of the partial products of the multiplication operation is optimized. Authors of [33] present more hardware details for reducing multiplication operation resources. They also mix the output from the PRNG with an auxiliary generator y^{t+1} to improve statistical tests.

Differential Chaotic PRNGs. These tools use an approximated numerical solution to solve a generalization of the Lörenz hyperchaos equation. The resolution was the main study done in [34] and in [35]. The authors design various numerical methods (Chen [36] and Elwakil) for each system. They show that obtained results with the Euler numerical approach are the best regarding area and throughput perspectives. Authors of [25], for their part, have implemented the so-called *Oscillator Frequency Dependent Negative Resistors* (OFDNR), and use the same Euler approximation.

The three best chaotic generators for FPGA appear to be, namely, the one from [27] that uses the logistic map with Matlab simulink macros, the chaotic iterations based PRNG [37], [38], and the one based on the chaotic *Bernoulli* map [30]. If we consider the linear PRNGs who pass TestU01 (see section below), these PRNGS have the most reduced throughput due to their use of multiplications and their various dependencies. However, to have a large throughput does not mean to produce an uniform distribution of numbers, which leads to the investigation of statistical results.

Finally, statistical tests are fast methods to study in practice the randomness of generated numbers, by the mean of software batteries of tests. They are based on various mathematical and physical approaches, and are thus used as generator benchmarks. To perform comparisons, in this study, we considered the reputed NIST $SP800-22$ [39] (10^6 pseudorandom bits evaluated by 16 tests) and TestU01 [23] batteries of tests (up to 10^{38} pseudorandom bits under 319 tests). According to [37] and [40], it can be observed that almost all chaotic PRNGs can pass the NIST batteries, but they all fail on TestU01. Meanwhile, a first application of chaotic iteration as PRNG approach was presented in two CIPRNG variants for FPGA, namely the CIPRNG-XOR and the CIPRNG-MultiCycle (see [37] [41]). This is why the works in [27] and in [30], based on the logistic map and the Bernoulli one, will be used for throughput comparison, while linear PRNGs will be considered for statistical tests. We can however already conclude that only XOR-CIPRNG satisfies both low hardware resources and a success against the TestU01 battery, which has already been stated in [37].

III. THE PROPOSAL

A. General idea

Formally speaking, a Chaotic Iteration based PRNG (CIPRNG) is a random walk in the graph of iterations of a specific binary function. The direction to take and the path length are defined by the embedded generator(s) [42]. A first application of such an approach was presented in the

PRNG framework [37], [41]. Meanwhile, in [43], the authors have proposed to remove an *Hamilton Cycle*, satisfying some balance properties, from the Markov chain on the N-cube, while in [21], authors proposed new functions without an Hamilton cycle, and studied the length of the walk in their cube, until having an associated *Markov* graph close enough to the uniform distribution. These works end with the idea that it is hard to have together the three properties of: chaos, hardware efficiency, and statistically trustworthy.

Let us first discuss on how we tackle this problem. The first key idea is to have a short internal state, possibly split into parallel blocs. This divide and conquer approach aims at ensuring hardware efficiency but is in conflict with statistical quality. Chaotic iterations [44], [45] can be used to achieve chaos objectives. However, as noticed in [21] the general formulation of the chaotic iterations [46] should be preferred than the original one when efficiency is needed. Finally, permutation techniques [47] have presented a convenient way to ensure statistical faultless, in a fast manner. Our proposal is based on these three main ideas and is summarized in Figure 1.

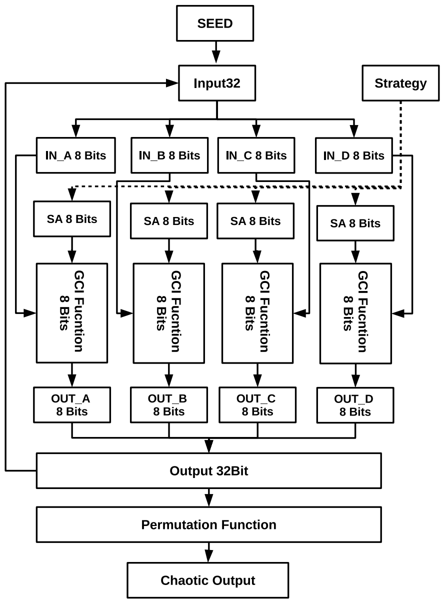

Fig. 1: The proposal based on GCI functions issued from removing a Hamilton cycle in the N-cube with a permutation function

At first, it can be seen that the seed x^0, the internal state x^t, and the output x^{t+1} are all expressed with the same number N of bits. Without loss of generality, we consider hereafter that $\mathsf{N} = 32$. Let us show how to produce a new output x^{t+1} for a given input x^t. This one is first split into n blocs of equal length. We consider here that $n = 4$ and we thus have $x^t = (x^t_A, x^t_B, x^t_C, x^t_D)$ where x^t_l is of size 8 for $l \in \{A,B,C,D\}$. The next step consists in obtaining a N bits number s^t from the embedded generator, which is called *the strategy*. Similarly to x^t, the vector s^t is split into n blocs. Here we thus obtain $s^t = (s^t_A, s^t_B, s^t_C, s^t_D)$. Each s_l, $l \in \{A,B,C,D\}$, can be interpreted as a set of elements in $\{1,2,\dots,8\}$. Each block x^t_l is modified separately as the result of the general formulation of the Chaotic Iterations [46] applied on x^t_l, s^t_l and a specific GCI function $f : \mathbb{B}^8 \to \mathbb{B}^8$, as described hereafter. The i-th component of x^{t+1}_l is the i-th one of $f(x^t_l)$ if i is within the set s^t_l, else this component is the i-th one of x^t_l (*i.e.*, only the components indicated by the set s^t_l are updated). This results x^{t+1}_l. More formally, we have

$$x^{t+1} = (x_1^{t+1},\dots,x_{\mathsf{N}}^{t+1}),\ 1 \le i \le \mathsf{N}, x_i^{t+1} = \begin{cases} f_i(x^t) & \text{if } i \in s^t, \\ x_i^t & \text{otherwise.} \end{cases}$$

All the x^{t+1}_l are concatenated hereafter, producing the new internal state x^{t+1}. Finally, a permutation over the N bits is applied on x^{t+1} to produce the new output.

The choice of the function f executed inside the GCI iteration, of the embedded PRNG, and of the chosen final permutation function has a great influence on the quality of the generator. It is discussed in the next sections.

B. Iterated Function

Let $s \in (\mathbb{B}^8)^{\mathbb{N}}$ be a sequence of subsets of $\{1,\dots 8\}$, where x^0 be a vector in $\mathbb{B}^8$ and f be a function from $\mathbb{B}^8 \to \mathbb{B}^8$.

Five functions from $\mathbb{B}^8$ to $\mathbb{B}^8$ are mainly studied in this article. The former is the negation function, further denoted as NG. In this one, each f_i is defined with $f_i(x) = \overline{x_i}$. For instance, the image of 5 = 00000101 is 250 = 11111010. The four other functions, denoted as F1, F2, F3, and F4, are the GCI functions whose graph of generalized iterations is strongly connected and which has been obtained by removing a balanced Hamiltonian cycle in a N-cube following the method suggested in [21] and refined in [48]. The choice of these five functions is motivated by the objective to obtain a chaotic behavior. For instance, it has been proven in [49] that the topological entropy of the chaotic iterations embedding the NG function is equal to $\log(8)$ when the iteration vector is constituted by 8 components. For the links between entropy and noise-like randomness, see, *e.g.*, [50].

C. Permutation Function

First of all, our proposal is a parallel execution of 4 blocks, each one producing 8 bits. The internal state x is next produced as the concatenation of the results of the 4 blocks. This design is guided by the goal of reducing the required resources. However, such an approach suffers from decreasing the statistical complexity of the PRNG: without any post treatment it would be dramatic, because it is equivalent to deal with 8 bits only. A final step which scrambles the internal state is thus necessary to tackle this problem.

This can be practically implemented with a permutation function (which allows to obtain a uniform output) provided it does not break the chaos property (as proven in the next section). Among the large choice of permutation functions (such as rotation, dropping, xoring...), we inspire from one detailed in [47]. This work indeed proposes a bench of permutation functions allowing to succeed statistical tests.

This permutation function is implemented as in Algorithms 1. It is not hard to see that it is mainly a composition of three subfunctions. Let *In32* be the internal state. The first function scrambles between 17 and 28 rightmost bits (*i.e.*

TABLE I: Boolean functions

Function	$f(x)$ for $x \in [0,1,2,3,4,5\ldots,2^n-1]$
NEG	[255, 254, 253, 252, 251, 250 …, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
F1	[223, 190, 249, 236, 243, 234, 241, 252, 183, 244, 229, 245, 179, 178, 225, 248, 237, 254, 173, 232, 171, 202, 201, 200, 247, 198, 228, 230, 195, 242, 233, 160, 215, 220, 205, 216, 218, 154, 221, 208, 213, 210, 212, 148, 147, 211, 217, 209, 239, 238, 141, 140, 235, 203, 193, 204, 135, 134, 199, 197, 131, 226, 129, 224, 63, 174, 253, 184, 251, 250, 189, 176, 191, 246, 180, 182, 51, 50, 185, 240, 47, 46, 175, 188, 139, 42, 161, 172, 231, 164, 181, 165, 227, 130, 33, 32, 31, 222, 153, 158, 219, 26, 25, 156, 159, 214, 151, 149, 146, 18, 144, 152, 207, 206, 157, 136, 138, 170, 169, 8, 133, 6, 5, 196, 3, 194, 137, 192, 255, 110, 109, 120, 107, 126, 125, 112, 103, 114, 116, 118, 123, 98, 121, 96, 79, 78, 111, 124, 75, 122, 97, 108, 71, 100, 117, 101, 115, 66, 113, 64, 127, 90, 89, 94, 83, 91, 81, 92, 95, 84, 87, 85, 82, 86, 80, 88, 77, 76, 93, 72, 74, 106, 105, 104, 69, 102, 68, 70, 99, 67, 73, 65, 55, 58, 57, 44, 187, 186, 49, 60, 119, 52, 37, 53, 35, 54, 177, 56, 45, 62, 61, 40, 59, 10, 9, 168, 167, 166, 36, 38, 163, 162, 41, 48, 23, 28, 13, 24, 155, 30, 29, 16, 21, 150, 20, 22, 27, 19, 145, 17, 143, 142, 15, 14, 43, 11, 1, 12, 39, 4, 7, 132, 2, 34, 0, 128]
F2	[223, 190, 249, 254, 243, 186, 233, 252, 183, 182, 247, 228, 242, 226, 240, 224, 237, 206, 173, 232, 203, 250, 169, 248, 167, 246, 245, 164, 235, 227, 241, 192, 215, 158, 157, 216, 218, 222, 221, 152, 213, 210, 149, 214, 219, 211, 217, 209, 239, 202, 207, 236, 139, 138, 193, 136, 231, 230, 199, 197, 194, 130, 225, 200, 63, 188, 253, 184, 251, 58, 189, 56, 191, 54, 165, 244, 51, 179, 161, 177, 47, 238, 175, 140, 163, 234, 41, 172, 39, 134, 229, 36, 162, 178, 129, 176, 31, 154, 29, 220, 147, 26, 145, 24, 159, 148, 151, 212, 146, 150, 144, 208, 141, 14, 205, 204, 171, 142, 201, 128, 133, 198, 132, 196, 195, 2, 137, 0, 255, 124, 109, 120, 122, 106, 125, 104, 117, 102, 101, 118, 123, 115, 97, 113, 79, 126, 111, 76, 99, 74, 121, 108, 71, 70, 103, 116, 98, 114, 65, 112, 127, 90, 89, 94, 83, 91, 81, 92, 95, 84, 87, 85, 82, 86, 80, 88, 77, 110, 93, 72, 107, 78, 105, 64, 69, 66, 68, 100, 75, 67, 73, 96, 55, 46, 57, 62, 187, 59, 185, 60, 119, 52, 181, 180, 50, 34, 48, 32, 45, 174, 61, 40, 11, 170, 9, 168, 37, 166, 53, 4, 43, 35, 49, 160, 23, 28, 13, 156, 155, 30, 153, 16, 21, 18, 20, 22, 27, 19, 25, 17, 143, 10, 15, 44, 3, 42, 1, 12, 135, 38, 7, 5, 131, 6, 33, 8]
F3	[223, 238, 189, 254, 243, 251, 233, 184, 183, 230, 229, 245, 242, 246, 177, 224, 237, 174, 253, 204, 203, 170, 201, 248, 247, 226, 197, 164, 235, 227, 241, 192, 215, 158, 205, 216, 155, 222, 221, 208, 151, 210, 212, 214, 219, 211, 145, 209, 143, 202, 207, 206, 139, 234, 193, 232, 135, 134, 199, 228, 194, 198, 129, 200, 63, 188, 61, 252, 186, 250, 249, 168, 191, 178, 180, 244, 187, 179, 49, 240, 239, 46, 175, 236, 163, 138, 185, 136, 231, 38, 181, 36, 162, 166, 225, 176, 31, 30, 153, 220, 147, 218, 217, 24, 159, 148, 213, 149, 19, 150, 144, 152, 141, 140, 13, 12, 171, 142, 9, 8, 133, 130, 5, 196, 195, 2, 137, 160, 255, 124, 109, 120, 122, 106, 125, 104, 103, 114, 116, 100, 123, 115, 97, 113, 79, 126, 111, 110, 99, 74, 121, 72, 71, 118, 117, 68, 98, 102, 65, 112, 127, 90, 89, 94, 83, 91, 81, 92, 95, 84, 87, 85, 82, 86, 80, 88, 77, 76, 93, 108, 107, 78, 105, 64, 69, 66, 101, 70, 75, 67, 73, 96, 55, 190, 57, 62, 51, 59, 41, 60, 119, 182, 37, 53, 50, 54, 48, 32, 45, 44, 173, 172, 11, 58, 169, 56, 167, 34, 165, 52, 43, 35, 161, 0, 23, 28, 157, 156, 26, 154, 29, 16, 21, 18, 20, 22, 27, 146, 25, 17, 47, 10, 15, 14, 3, 42, 1, 40, 39, 4, 7, 132, 131, 6, 33, 128]
F4	[223, 250, 249, 254, 243, 234, 185, 232, 183, 244, 229, 180, 242, 178, 240, 248, 237, 206, 253, 252, 171, 170, 201, 224, 247, 246, 165, 230, 195, 227, 161, 192, 215, 220, 205, 216, 218, 222, 153, 208, 151, 150, 212, 214, 219, 211, 217, 209, 239, 202, 207, 236, 235, 138, 137, 204, 135, 196, 199, 228, 194, 130, 225, 128, 63, 188, 61, 172, 251, 190, 189, 176, 191, 166, 245, 182, 187, 50, 241, 177, 143, 238, 175, 140, 43, 42, 233, 184, 231, 164, 37, 132, 35, 226, 33, 168, 31, 154, 221, 158, 27,155, 145, 156, 159, 22, 213, 149, 146, 210, 144, 152, 141, 14, 157, 136, 203, 142, 9, 200, 7, 198, 197, 4, 163, 131, 193, 0, 255, 124, 109, 108, 107, 126, 125, 112, 103, 102, 116, 118, 123, 115, 97, 113, 79, 106, 111, 76, 75, 122, 121, 120, 71, 100, 117, 68, 98, 114, 65, 104, 127, 90, 89, 94, 83, 91, 81, 92, 95, 84, 87, 85, 82, 86, 80, 88, 77, 110, 93, 72, 74, 78, 105, 64, 69, 66, 101, 70, 99, 67, 73, 96, 55, 58, 57, 62, 51, 186, 41, 40, 119, 52, 181, 53, 179, 34, 48, 56, 45, 174, 173, 60, 59, 46, 169, 32, 167, 54, 5, 38, 3, 162, 49, 160, 23, 28, 13, 24, 26, 30, 29, 16, 21, 18, 20, 148, 147, 19, 25, 17, 47, 10, 15, 44, 139, 11, 1, 12, 39, 134, 133, 36, 2, 6, 129, 8]

Algorithm 1 Random Xorshift Permutation Function

```
Input: $In32$ (32 bits word); Output: $Out32$ (a 32 bits word)
1: $word1 \leftarrow (In32 \gg ((In32 \gg 28u) + 4u)) \otimes In32$,
2: $word2 \leftarrow word1 * b$,
3: $word3 \leftarrow (word2 \gg 22u) \otimes word2$,
4: return $Out32 \leftarrow word3$.
```

middle bits) with a xor function. The number of selected elements depends on the value of $In32$. Then, the second function applies a modular multiplication in the cyclic group of elements in $\{1,\ldots,2^{31}-2\}$. The chosen multiplier b is a primitive root of the modulus $2^{31}-1$ and in this work is set to 277803737. The latter function is a simple right xorshift on the lowest bits to scramble them.

The next section proves that such a proposal provides a chaotic PRNG.

IV. MATHEMATICAL CHAOS OF THE PROPOSED DESIGN

A. *First considerations*

We want now to characterize how much chaotic is our proposal. By chaos, we mean that the effects, on the output sequence, of any slight alteration of either the seed or the embedded PRNG cannot be predicted in the short or medium term. Additionally, with two different strategies or seeds, we can reach twice the same x^t, but with two different x^{t+1}: this finite state machine does not necessarily enter into a loop, as at each iteration we take a new value from the "outside world", that is, from the strategy (this machine does not iterate in a vacuum). Furthermore, this strategy can be non-periodic, if we consider a TRNG like a physical white noise or any physical source of entropy of that kind.

To sum up, by designing a finite state machine that only manipulate integers (no floating point issue), but which takes a new input from the outside world at each iteration, we thus obtain an iterative process working on an infinite space, and which does not enter necessarily within a loop. We are then left to evaluate the chaotic behavior of the proposal depicted in Figure 1.

B. *Proof of chaos: the internal process*

Let us first specify some notations and definitions in use in this section. In what follows, $\mathbb{B}$ is the Boolean set, while $\mathbb{R}$ and $\mathbb{N}$ are the usual sets of real and integer numbers, with the notations of $\mathbb{N}^*$ and $\mathbb{R}_+$ for their positive (or equal to zero) subsets. For $a,b \in \mathbb{N}$, $[\![a,b]\!]$ is the set of integers: $\{a,a+1,\ldots,b\}$. $X^{\mathbb{N}}$ is the set of sequences belonging in X and s^k is the k-th term of a sequence $s=\left(s^k\right)_{k\in\mathbb{N}}$, which may be a vector. Finally, f^n means the n-th composition of the function f (*i.e.*, $f^n = f \circ f \circ \ldots \circ f$), while v_i is the i-th component of a vector v.

Consider a topological space $(\mathcal{X},\tau)$ and a continuous function $f : \mathcal{X} \to \mathcal{X}$. A discrete dynamical system is said chaotic [51] when it satisfies the three following properties defined by Devaney [52]: *Transitivity*, *Regularity*, and *Sensibility to the initial conditions* [53]. Note that various other mathematical definitions of chaos, or of "unpredictability" for a dynamical system, can be found in the literature, like Li-Yorke and Knudsen chaos, mixing, topological and metrical positive entropy. They all are non equivalent and complementary, but the Devaney's formulation is usually the first one to state.

We first focus on the proposal without the permutation, which is referred as the internal process.

For $\mathsf{N} \in \mathbb{N}^*$, let us define the set

$$\mathcal{X}_{\mathsf{N}} = \mathbb{B}^{\mathsf{N}} \times \left(\mathbb{B}^{\mathsf{N}}\right)^{\mathbb{N}},$$

and the following functions:

$$\begin{array}{cccc} i_{\mathsf{N}}: & \left(\mathbb{B}^{\mathsf{N}}\right)^{\mathbb{N}} & \longrightarrow & \mathbb{B}^{\mathsf{N}} \\ & (s^k)_{k\in\mathbb{N}} & \longmapsto & s^0, \end{array}$$

and

$$\begin{array}{cccc} \sigma_{\mathsf{N}}: & \left(\mathbb{B}^{\mathsf{N}}\right)^{\mathbb{N}} & \longrightarrow & \left(\mathbb{B}^{\mathsf{N}}\right)^{\mathbb{N}} \\ & (s^k)_{k\in\mathbb{N}} & \longmapsto & (s^{k+1})_{k\in\mathbb{N}}. \end{array}$$

They respectively extracts the first term of an inputted sequence (i_{N}), and shifts it to the left by removing the first term (σ_{N}).

We now define the distances:

$$\begin{array}{cccc} d_{\mathsf{N},E}: & \mathbb{B}^{\mathsf{N}} \times \mathbb{B}^{\mathsf{N}} & \longrightarrow & \mathbb{R}_+ \\ & (e,e') & \longmapsto & \sum_{k=1}^{\mathsf{N}} |e_k - e'_k|, \end{array}$$

which is the Hamming distance on $\mathbb{B}^{\mathsf{N}}$, and

$$\begin{array}{cccc} d_{\mathsf{N},S}: & \left(\mathbb{B}^{\mathsf{N}}\right)^{\mathbb{N}} \times \left(\mathbb{B}^{\mathsf{N}}\right)^{\mathbb{N}} & \longrightarrow & \mathbb{R}_+ \\ & (s,s') & \longmapsto & \dfrac{9}{\mathsf{N}} \sum_{k=0}^{\infty} \dfrac{d_{\mathsf{N},E}(s^k,s'^k)}{10^{k+1}}. \end{array}$$

It has been already proven in [46] (by identifying, *mutatis mutandis*, the set of subsets of $[\![1,\mathsf{N}]\!]$ with $\mathbb{B}^{\mathsf{N}}$) that, with the distance $d_{\mathsf{N}} = d_{\mathsf{N},E} + d_{\mathsf{N},S}$, $\mathcal{X}_{\mathsf{N}}$ becomes a metric space. We define

$$\begin{array}{rccc} F_{\mathsf{N},f}: & \mathbb{B}^{\mathsf{N}} \times \mathbb{B}^{\mathsf{N}} & \longrightarrow & \mathbb{B}^{\mathsf{N}} \\ & (b,e) & \longmapsto & F_{\mathsf{N},f}(b,e), \end{array}$$

where $\forall i \in [\![1,\mathsf{N}]\!]$,

$$F_{\mathsf{N},f}(b,e)_i = \begin{cases} e_i & \text{if } b_i = 0, \\ f(e)_i & \text{else,} \end{cases}$$

and

$$\begin{array}{rccc} g_f: & \mathcal{X}_8 & \longrightarrow & \mathcal{X}_8 \\ & (e,s) & \longmapsto & (F_{8,f}(i_8(s),e);\sigma_8(s)). \end{array}$$

It has already been established, in [46], that such general iterations are continuous on the metric space $(\mathcal{X}_8, d_8)$, and that the discrete dynamical space $x^0 \in \mathcal{X}_8$, $x^{n+1} = g_f(x^n)$ is chaotic, according to Devaney, on $(\mathcal{X}_8, d_8)$. It is shown too that this dynamical system is strongly transitive [46].

Given $n, N \in \mathbb{N}, N \geqslant n$, we define:

$$\begin{array}{rccc} \psi_{n,N}: & [\![1, N-n+1]\!] \times \mathbb{B}^N & \longrightarrow & \mathbb{B}^n \\ & (k,e) & \longmapsto & (e_k, \ldots, e_{k+n-1}), \end{array}$$

and, similarly, $\Psi_{n,N}$, as follows:

$$\begin{array}{ccc} [\![1, N-n+1]\!] \times \left(\mathbb{B}^N\right)^{\mathbb{N}} & \longrightarrow & (\mathbb{B}^n)^{\mathbb{N}} \\ (k, (s^i)_{i\in\mathbb{N}}) & \longmapsto & (\Psi_{n,N}(S_i))_{i\in\mathbb{N}}, \end{array}$$

and, finally,

$$\begin{array}{rccl} h: & \mathcal{X}_{32} & \longrightarrow & \mathcal{X}_{32} \\ & (e,s) & \longmapsto & \big((g_f(\psi_{8,32}(1,e),\Psi_{8,32}(1,s))_{1,1}, \\ & & & \vdots \\ & & & g_f(\psi_{8,32}(1,e),\Psi_{8,32}(1,s))_{1,8}, \\ & & & g_f(\psi_{8,32}(9,e),\Psi_{8,32}(9,s))_{1,1}, \\ & & & \vdots \\ & & & g_f(\psi_{8,32}(9,e),\Psi_{8,32}(9,s))_{1,8}, \\ & & & g_f(\psi_{8,32}(17,e),\Psi_{8,32}(17,s))_{1,1}, \\ & & & \vdots \\ & & & g_f(\psi_{8,32}(17,e),\Psi_{8,32}(17,s))_{1,8}, \\ & & & g_f(\psi_{8,32}(25,e),\Psi_{8,32}(25,s))_{1,1}, \\ & & & \vdots \\ & & & g_f(\psi_{8,32}(25,e),\Psi_{8,32}(25,s))_{1,8}), \\ & & & \sigma_{32}(s)\big). \end{array}$$

We can remark that the h function is what is iterated inside our proposal, if we except the permutation. Indeed, the four blocks (binary digits ranging from 1 to 8, and then from 9 to 16, from 17 to 24, and finally from 25 to 32) appear well in the first component of the output of h.

We will show that iterations of h are chaotic on $\mathcal{X}_{32}$ and, using a topological semi-conjugacy, that the permutation does not alter such an unpredictable behavior. In order to do so, we must first check that,

Proposition 1. *h is a continuous map on the metric space $(\mathcal{X}_{32}, d_{32})$.*

Proof. Let us consider a sequence $x^n = (e^n, s^n)_{n\in\mathbb{N}} \in \mathcal{X}_{32}^{\mathbb{N}}$, which is convergent to an element $x = (e,s) \in \mathcal{X}_{32}^{\mathbb{N}}$. As

$$\begin{aligned} d_{32}(x^n,x) &\longrightarrow 0 \\ &= d_{32,E}(e^n,e) + d_{32,S}(s^n,s), \end{aligned}$$

and due to the fact that $d_{32,E}$ produces only integers, we have $\exists n_1 \in \mathbb{N}, n \geqslant n_1 \Rightarrow e^n = e$.

Similarly, $d_{32,S}(s^n,s) \longrightarrow 0$, so $\exists n_2 \in \mathbb{N}$ such that $n \geqslant n_2 \Rightarrow d_{32,S}(s^n,s) < \frac{1}{10^{32}}$. Due to the way we defined $d_{\mathsf{N},S}$, we can conclude that $\forall n \geqslant n_2$, the sequence s^n has the same first 32 terms than the sequence s. And we can conclude from these two facts that

$$\forall n \geqslant \max\{n_1,n_2\}, h(e^n,s^n)_1 = h(e,s)_1.$$

Finally, for $n \geqslant n_2$, we have $\forall i < 32$, $s^{n,i} = s^i$. So, $\forall n \geqslant n_2$,

$$\begin{aligned} d_{32,S}(s^n,s) &= \frac{9}{\mathsf{N}} \sum_{k=0}^{\infty} \frac{d_{\mathsf{N},E}(s^{n,k},s^k)}{10^{k+1}} \\ &= \frac{9}{\mathsf{N}} \sum_{k=0}^{\infty} \frac{d_{\mathsf{N},E}(\sigma(s^n)^k,\sigma(s)^k)}{10^{k+1}} \\ &= \frac{d_{32,S}(\sigma_{32}(s^n),\sigma_{32}(s))}{10}. \end{aligned}$$

As $d_{32,S}(s^n,s) \longrightarrow 0$, we can deduce that $d_{32,S}(\sigma_{32}(s^n),\sigma_{32}(s)) \longrightarrow 0$, and so:

$$h(e^n,s^n)_2 \longrightarrow h(e,s)_2.$$

As a conclusion, for all sequence $x^n = (e^n,s^n)_{n\in\mathbb{N}}$ of $\mathcal{X}_{32}^{\mathbb{N}}$, if $x^n \longrightarrow x = (e,s) \in \mathcal{X}_{32}$, then $h(x^n) \longrightarrow h(x)$. This is the sequential characterization of the continuity, and so h is continuous on $(\mathcal{X}_{32}, d_{32})$. □

Let us now show that:

Proposition 2. *If g_f is strongly transitive on $\mathcal{X}_8$, then h_f is chaotic according to Devaney on $(\mathcal{X}_{32}, d_{32})$.*

Proof. Let us first prove that,

Lemma 1. *If g_f is strongly transitive on $\mathcal{X}_8$, then h_f is strongly transitive on $(\mathcal{X}_{32}, d_{32})$.*

Proof. Let $x = (e,s)$ and $\check{x} = (\check{e},\check{s})$ two points of $\mathcal{X}_{32}$, and $\varepsilon > 0$. We must find $x' = (e',s')$ inside the open ball $\mathcal{B}(x,\varepsilon) = \{u \in \mathcal{X}_{32}, d_{32}(u,x) < \varepsilon\}$ such that:

$$h_f^n(x') = \check{x}.$$

Let us consider:

$$\begin{aligned} p_1 = (\psi_{8,32}(1,e);(& \Psi_{8,32}(1,s),\Psi_{8,32}(32+1,s), \\ & \Psi_{8,32}(2\times 32+1,s),\ldots, \\ & \Psi_{8,32}(k\times 32+1,s),\ldots)), \\ q_1 = (\psi_{8,32}(1,\check{e});(& \Psi_{8,32}(1,\check{s}),\Psi_{8,32}(32+1,\check{s}), \\ & \Psi_{8,32}(2\times 32+1,\check{s}),\ldots, \\ & \Psi_{8,32}(k\times 32+1,\check{s}),\ldots)), \end{aligned}$$

in which the second components are infinite sequences of $\mathbb{B}^8$. p_1 and q_1 belong to $\mathcal{X}_8$ and g_f is strongly transitive, so there exist $\widetilde{p_1} = ((\widetilde{e_1}, \dots \widetilde{e_8}), (\widetilde{s_1}, \widetilde{s_2}, \dots))$ in $\mathcal{B}(p_1, \varepsilon)$ and $n_1 \in \mathbb{N}$ such that:

$$g_f^{n_1}(\widetilde{p_1}) = q_1.$$

We can apply the same process on points:

$$\begin{aligned} p_2 = (\psi_{8,32}(9,e);(\quad & \Psi_{8,32}(9,s), \Psi_{8,32}(32+9,s), \\ & \Psi_{8,32}(2\times 32+9,s), \dots, \\ & \Psi_{8,32}(k\times 32+9,s), \dots)), \\ q_2 = (\psi_{8,32}(9,\check{e});(\quad & \Psi_{8,32}(9,\check{s}), \Psi_{8,32}(32+9,\check{s}), \\ & \Psi_{8,32}(2\times 32+9,\check{s}), \dots, \\ & \Psi_{8,32}(k\times 32+9,\check{s}), \dots)), \end{aligned}$$

leading to the existence of $\widetilde{p_2} \in \mathcal{X}_8$ and $n_2 \in \mathbb{N}$ such that $g_f^{n_2}(\widetilde{p_2}) = q_2$. The process if finally applied on the last two "quarters" of $\mathcal{X}_{32} = \mathbb{B}^{32} \times \left(\mathbb{B}^{32}\right)^{\mathbb{N}}$, dividing the first (resp. second) set of the Cartesian product in 4 vectors of 8 bits (resp. in sequences belonging in $\mathbb{B}^8$) thanks to $\phi_{8,32}$ (resp. $\Phi_{8,32}$). This leads to the points of $\mathcal{X}_8$ defined below:

$$\begin{aligned} p_3 = (\psi_{8,32}(17,e);(\quad & \Psi_{8,32}(17,s), \Psi_{8,32}(32+17,s), \\ & \Psi_{8,32}(2\times 32+17,s), \dots)), \\ q_3 = (\psi_{8,32}(17,\check{e});(\quad & \Psi_{8,32}(17,\check{s}), \Psi_{8,32}(32+17,\check{s}), \\ & \Psi_{8,32}(2\times 32+17,\check{s}), \dots)), \\ p_4 = (\psi_{8,32}(25,e);(\quad & \Psi_{8,32}(25,s), \Psi_{8,32}(32+25,s), \\ & \Psi_{8,32}(2\times 32+25,s), \dots)), \\ q_4 = (\psi_{8,32}(25,\check{e});(\quad & \Psi_{8,32}(25,\check{s}), \Psi_{8,32}(32+25,\check{s}), \\ & \Psi_{8,32}(2\times 32+25,\check{s}), \dots)). \end{aligned}$$

As previously, due to the strong transitivity of g_f, we have the existence of $\widetilde{p_3}, \widetilde{p_4} \in \mathcal{X}_8$ and of $n_3, n_4 \in \mathbb{N}$ such that $g_f^{n_3}(\widetilde{p_3}) = q_3$ and $g_f^{n_4}(\widetilde{p_4}) = q_4$.

Let us introduce the following notation: $\widetilde{p_i} = (\widetilde{e_i}, \widetilde{S_1})$ for $i = 1, \dots, 4$, and $n_0 = \max_{i=1}^{4}\{n_i\}$. We define:

$$\widetilde{s_i}^k = \begin{cases} S_i^k & \text{if } \ k \leqslant n_i, \\ 0 & \text{if } \ k \in [\![n_i+1, n_0]\!], \\ S_i^{k-n_0+n_i} & \text{else.} \end{cases}$$

Indeed, for each quarter of $\mathcal{X}_{32}$ we have four different $n_i, i = 1..4$, number of iterations to reach a given point of $\mathcal{X}_8$ by starting to a neighborhood of another given point of this quarter. By adding 0's in the iteration sequence, we thus allow to iterate in a vacuum the required number of times in each quartet, so that after n_0 iterations each 4 part of the targeted point x' are reached. Let us do it with details.

Let us consider the point $p' = (e', s') \in \mathcal{X}_{32}$ defined by:

- $e' = (\widetilde{e}_{1,1}, \dots, \widetilde{e}_{1,8}, \widetilde{e}_{2,1}, \dots, \widetilde{e}_{2,8}, \widetilde{e}_{3,1}, \dots, \widetilde{e}_{3,8}, \widetilde{e}_{4,1}, \dots, \widetilde{e}_{4,8}) \in \mathbb{B}^{32}$,
- $s' = ((\widetilde{s}_{1,8k+1}, \dots, \widetilde{s}_{1,8k+8}, \widetilde{s}_{2,8k+1}, \dots, \widetilde{s}_{2,8k+8}, \widetilde{s}_{3,8k+1}, \dots, \widetilde{s}_{3,8k+8}, \widetilde{s}_{4,8k+1}, \dots, \widetilde{s}_{4,8k+8}))_{k\in\mathbb{N}}$, which is a sequence of $\left(\mathbb{B}^{32}\right)^{\mathbb{N}}$.

By construction, this point of $\mathcal{X}_{32}$ is such that $h_f^{n_0}(x') = \check{x}$ and $x' \in \mathcal{B}(x, \varepsilon)$. □

Let us now finalize the proof of Prop. 2. h_f being strongly transitive on $(\mathcal{X}_{32}, d_{32})$, it is thus transitive. We are then left to establish the regularity of h_f.

Let us consider $x = (e, s) \in \mathcal{X}_{32}$, and $\varepsilon > 0$. We need to find a periodic point $x' = (e', s')$ inside $\mathcal{B}(x, \varepsilon)$. As ε may be lower than 1, and due to the definition of d_{32}, we must choose $e' = e$. Let $k_0 = -\lfloor \log_{10}(\varepsilon) \rfloor + 1$ the integer such that any point of the form $(e, (s^0, s^1, \dots, s^{k_0}, a, b, c, \dots))$ is inside $\mathcal{B}(x, \varepsilon)$.

Let us denote by $\check{x} = (\check{e}, \check{s})$ the point $h_f^{k_0}(x)$. Due to the strong transitivity of h_f (Lemma 1), there is a point $\widetilde{x} = (\widetilde{e}, \widetilde{s})$ in $\mathcal{B}(\check{x}, 0.1)$ and $k_1 \in \mathbb{N}$ such that $h_f^{k_1}(\widetilde{x}) = x$. Finally, the point $x' = (e, (s^0, s^1, \dots, s^{k_0}, \widetilde{s}^0, \dots, \widetilde{s}^{k_1}, s^0, s^1, \dots, s^{k_0}, \widetilde{s}^0, \dots, \widetilde{s}^{k_1}, \dots)$ is $k_0 + k_1$ periodic and inside the neighborhood $\mathcal{B}(x, \varepsilon)$ of x, which proves the regularity, and then the chaotic behavior of h_f. □

C. *Proof of chaos: the whole generator*

In the proposal, the internal function h_f is iterated on the current internal state, and with a and with a new generated sequence taken from the outer strategy. Then, the output is a permutation p of the internal state, which is not internally modified. To prove that the whole generator G_f is chaotic, this permutation p must be integrated inside the iterations, to see if the output has a chaotic behavior when modifying the input (internal state or strategy). To write the generator as a discrete dynamical system, we need to introduce the reverse permutation p^{-1}.

Let us define

$$\begin{array}{cccc} \mathsf{P}: & \mathcal{X}_{32} & \longrightarrow & \mathcal{X}_{32} \\ & (e,s) & \longmapsto & (\mathsf{p}(e), s), \end{array}$$

its inverse being

$$\begin{array}{cccc} \mathsf{P}^{-1}: & \mathcal{X}_{32} & \longrightarrow & \mathcal{X}_{32} \\ & (e,s) & \longmapsto & (\mathsf{p}^{-1}(e), s). \end{array}$$

We can now introduce the following diagram:

$$\begin{tikzcd} \mathcal{X}_{32} \arrow[r, "h_f"] & \mathcal{X}_{32} \arrow[d, "\mathsf{P}"'] \\ \mathcal{X}_{32} \arrow[u, "\mathsf{P}^{-1}"'] \arrow[r, "G_f"] & \mathcal{X}_{32} \end{tikzcd}$$

P^{-1} and P are obviously continuous on $(\mathcal{X}_{32}, d_{32})$, which can be directly deduced by the sequential characterization of the continuity. So the commutative diagram depicted above is a topological conjugacy, and the generator

$$G_f = \mathsf{P} \circ h_f \circ \mathsf{P}^{-1},$$

thus inherits the chaotic behavior of h_f on $(\mathcal{X}_{32}, d_{32})$.

V. Cryptographic Analysis

After having investigated the chaos properties of our generator, we now discuss about security.

Definition V.1. A cryptographic PRNG (cPRNG) is a deterministic algorithm G transforming strings into strings and such that, for any seed s of length m, $G(s)$ (the output of G on the input s) has size $l_G(m)$ with $l_G(m) > m$.

Definition V.2. A cPRNG G is *secure* if for any probabilistic polynomial time algorithm D, for any positive polynomial p, and for all sufficiently large m s,

$$|\mathbb{P}[D(G(\delta_m))=1]-\mathbb{P}[D(\delta_{l_G(m)})=1]|<\frac{1}{p(m)},$$

where δ_m is the uniform distribution on $\mathbb{B}^m$ and the probabilities $\mathbb{P}$ are taken over δ_m, $\delta_{l_{G(m)}}$ as well as over the internal coin tosses of D.

Intuitively, it means that there is no polynomial time algorithm that can distinguish a perfect uniform random generator from G with a non negligible probability.

Now fix a seed and suppose that the strategy in our generator is computed by a secure cPRNG, say G', then the whole process, say G, is secure too. Indeed it has been shown that:

Proposition 3. Let $f:\mathbb{B}^n\to\mathbb{B}^n$, $\Gamma(f)$ its iteration graph, $\check{M}$ its adjacency matrix and M a $n\times n$ matrix defined by $M_{ij}=\frac{1}{n}\check{M}_{ij}$ if $i\neq j$ and $M_{ii}=1-\frac{1}{n}\sum\limits_{j=1,j\neq i}^{n}\check{M}_{ij}$ otherwise.

If $\Gamma(f)$ is strongly connected, then the output of $GCIPRNG_f$ follows a law that tends to the uniform distribution if and only if M is a double stochastic matrix.

With this result, is not hard to see that the proposed algorithm preserves the security property.

Proposition 4. If in the algorithm described in Figure 1 the strategy is computed by a secure cPRNGs G', then for any fixed seed the whole algorithm is secure (with respect to the input of G').

Proof. Suppose that G' takes in input m bits and returns $4m$ bit (thus $l_{G'}(m)=4m$); since G' is secure, for any probabilistic polynomial time algorithm D, for any positive polynomial p, and for all sufficiently large m s,

$$|\mathbb{P}[D(G'(\delta_m))=1]-\mathbb{P}[D(\delta_{4m})=1]|<\frac{1}{p(m)}.$$

Let us decompose the algorithm in Figure 1 in three parts $G=P\circ C\circ G'$ *i.e.*, we firstly produce a strategy, then we apply the chaotic iterations C as described before and finally a permutation function P.

Fix a probabilistic polynomial time algorithm D and a polynomial p, our aim is to prove that for any sufficiently large m

$$|\mathbb{P}[D(G(\delta_m))=1]-\mathbb{P}[D(\delta_{4m})=1]|<\frac{1}{p(m)},$$

that is equivalent to

$$|\mathbb{P}[D(P(C(G'(\delta_m))))=1]-\mathbb{P}[D(\delta_{4m})=1]|<\frac{1}{p(m)}.$$

Since the CIs have a uniformed distributed output (in the sense of the proposition above)

$$\mathbb{P}[D(C(\delta_{4m}))=1]=\mathbb{P}[D(\delta_{4m})=1],$$

and since P is a bijective function we also have

$$\mathbb{P}[D(P(\delta_{4m}))=1]=\mathbb{P}[D(\delta_{4m})=1],$$

and thus the thesis can be rewritten as

$$|\mathbb{P}[D(P(C(G'(\delta_m))))=1]-\mathbb{P}[D(P(C(\delta_{4m})))=1]|<\frac{1}{p(m)}$$

,

and the formula above is true, because G' is secure for *any* probabilistic polynomial time algorithm, and so it is secure for $D\circ P\circ C$ too. □

At this point, we have at hand a tool that is both chaotic and cryptographically secure, thus achieving two objectives that are often both sought-after and rarely achieved in a rigorous manner [54].

VI. Hardware Implementation on FPGA

A. General Presentation

This new platform is an alternative hardware and test concept of the Zynq one that has been proposed in the research work [37]. Here, the platform is fully independent of any CPU (Zynq) and it is fully reconfigurable with a main software.

Figure 2 presents the main architecture of the AXI-test platform, which consists of the following components. A *Decoder Command Controller Unit* (DCCU), a *Design Under Test* (DUT) controller based on the GCIPRNG, and an *Universal Asynchronous Receiver Transmitter* (UART) serial port. All these units are compatible with the *AXI-4 Lite* bus protocol, which resumes the data transition and handshaking communication between units. Additionally, each of these units has an address map and an identifier (ID), which can be read and reconfigured. On the one hand, the DCCU decodes all commands received from both UART (PC-FPGA-PC) and DUT controller. It also chooses the strategy used in the GCIPRNG. Additionally, the DCCU defines latency of the final outputs, and the read&write operations in the internal registers of the platform (UART, strategy, and GCIPRNG that is tested). Finally, a software application is deployed with this platform, to have a full control and access to the GCIPRNGs tested in FPGA. Note that this AXI-test platform allows too the runs of TestU01 statistical tests in real time.

For the experiments, the test platform is designed and implemented using Xilinx Vivado tools and two FPGA prototype boards, namely the ZYBO board and Nexys V.4 Artix−7. The system is embedded with at least 3 strategies for the GCIPRNG core (unary, parallel, or generalized chaotic iterations), where the hardware resources are 2.5 times lower than in the Zynq platform [37], with 1211 LUT (1.19%), 1467 FF (1.16%), and 211Mhz respectively.

B. Global Comparison

Table II presents the results of two different implementations of our proposal on FPGA with their TestU01 statistical test evaluations. During these implementations, we considered five distinct Boolean functions, namely the negation and GCI F1, F2 as mentioned in Section III (F3 and F4 have the same behaviors than F1 and F2). To pass TestU01, the multiplier constant “b” of the permutation function (see Algorithm 1) must be equal to 811 for all strategies based 32 bits generators and 995 for 64 bits (as a comparison, we found 277803737

TABLE II: FPGA Implementation of 32 bits and 64 bits GCIPRNG using different linear PRNGs as strategies

	PRNG	32 bits GCIPRNG						64 bits GCIPRNG					
	Functions	Negation		F1		F2		Negation		F1		F2	
	Strategies	Taus88	LFSR113	Taus88	LFSR113	Taus88	LFSR113	Taus88+ LFSR113	xorshift128P64	Taus88+ LFSR113	xorshift128P64	Taus88+ LFSR113	xorshift128P64
AREA	LUT	263	273	421	433	421	415	617	662	874	914	881	906
	FF	305	344	412	518	412	444	740	740	748	785	748	785
	Total Area (LUT+FF)*8	4544	4936	6664	7608	6664	6872	10856	11216	12976	13592	13032	13528
SPEED	Frequency (Mhz)	199.3	217.3	200.68	197.32	194.93	205.76	180.15	172.5	177.03	171.71	176.09	169.22
	Output Latency	1	1	1	1	1	1	1	1	1	1	1	1
	Throughput/Latency (Gbps)	6.38	6.95	6.42	6.31	6.24	6.58	11.53	11.04	11.33	10.99	11.27	10.83
TEST	NIST	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
	TestU01	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS

TABLE III: ASIC implementation of GCIPRNG with other PRNGs Lightweight Primitives

PRNG	GCIPRNG With ermutation		GCIPRNG Without permutation		Other PRNGs lightweight primitives											
					PRNG						Stream chiper				TRNG	
	GCI-NEG with LFSR113	GCI-F1 with LFSR113	GCI-NEG with LFSR113	GCI-F1 with LFSR113	Warbler [55]	Warbler [56]	LAMED [57]	Melia-Segui [58]	J3Gen [59]	AKARI1B [60]	Grain [61]	Trivium [61]	Grain [62]	Trivium [62]	PVT [63]	[64]
Technology (*nm*)	65	65	65	65	65	65	***	***	130	90	130	130	65	65	45	180
Gate Elements (GE μm^2)	3584.72	4774,30	1709	2799,3	511	1238	1585	761	1190	1749	1259	2088	1126	1986	4004	**
Frequency (Mhz)	296	279	301	740	1370	689	***	***	***	***	***	***	1020	962	****	***
Output Latency	1	1	5	5	160	160	194	220	***	***	160	1152	***	***	32	***
Throughput (Gbps)	9.4	8.9	23.6 /5	9.65 /5	0.556	0.396	***	***	***	***	***	***	1.02	0.962	$2.4*10^{-6}$	$5*10^{-3}$
Total Power (mw)	1.8	2.36	0.6	1.04	***	5.83	2.04	***	***	***	***	***	2.04	3.88	7.0	3.6
Throughput@14Mhz (Kbps)	$3.2x10^6$	$3x10^6$	$17.8x10^6$ /5	$8.3x10^6/5$	***	***	***	***	***	***	***	***	***	***	***	***
Total Power@14Mhz (μW)	102.8	135.1	60.1	35.7	***	***	***	***	***	***	***	***	***	***	***	***
Throughput@100Khz (Kbps)	$4.3x10^6$	$3.2x10^6$	$16x10^6/5$	$11x10^6/5$	20	***	8.2	***	***	14.2	100	100	***	***	***	***
Total Power@100Khz (μW)	6.8	7.8	3.31	4.5	1.3	***	***	***	156.3 nW	0.182	0.78	1.44	2.04	3.88	***	***
NIST	PASS	PASS	PASS	PASS	PASS	PASS	PASS	NO	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
TestU01	PASS	PASS	PASS	PASS	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO

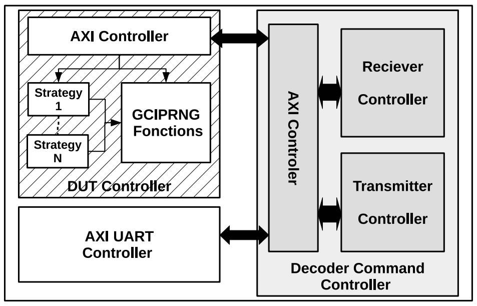

Fig. 2: FPGA Test platform for GCIPRNG

for PCG32). Linear PRNGs are used as strategies (inputted generators), which are LFSR113, Taus88, and xorshift128P64. All the design is synchronized with a main clock of $125Mhz$ and reset. Obtained results are described hereafter.

Negation Function. Three implementations have been realized and evaluated for each GCIPRNG width (32/64 bits). For 32 bits generators, it is obviously more efficient than its best competitors recalled in this paper, as its throughput is between 1 and 6 times larger than the other chaotic PRNGs (that cannot pass TestU01). However, the exception comes from [27] using the logistic map and Berouili [65]: it is true that the latter has a throughput of 7.5 and 8.5 Gbps for 32 bits (we discarded [27], as this latter is fully dependent on Matlab Simulink macros, which is not relevant for ASIC implementation). Similarly, our three implementations using the negation function exhibit less robust results compared to XOR-CIPRNG [37] for throughput compared to the area.

Finally, compared to the 32 bits PRNGs that can also pass TestU01, our 64 bits generators with the negation function use less area and are faster. To conclude this part, and when considering the negation, our proposal using LFSR113 as strategy is our best candidate for 32 bits generating 6.96 Gbps, which is increased to 11.5 Gbps for 64 bits generators.

GCI functions. We performed similar experiments than for the negation function. We obtained a lower performance in terms of throughout when compared with the negation function, which is due to the function implementation, and because in the negation we iterate a very simple logical operation (see Algorithm 1 to compare). However, despite its use of a bigger constant, which leads to a longer data path, the proposal with GCI functions does not consume any DSP block of FPGA: logic operators are sufficient. Additionally, results show that the three implementations with GCI functions perform better than all the other chaotic PRNGs that can pass the TestU01, if we except both our proposal with the negation and the XOR-CIPRNG [37]. Their performances are close to what has been obtained with the negation function, or to [37] with Taus88 as strategy, while GCIPRNG makes harder to reverse the process without knowing the internal transition function. However, XOR-CIPRNG [37] is limited to 32 bits (8.5 Gbps) and uses three different strategies (two 64 bits and one 32 bits), when GCIPRNG is up to 64 bits (≈ 11.5 Gbps) while less strategies are used (two 32 bits). Indeed, GCIPRNG is much practical for embedded cryptographical applications and system with a flexibility of integration in SoC for different sizes.

C. Statistical Tests Analysis

The test batteries have been run in Z-book Intel Core $i7-4800MQCPU@2.70GHz \times 8$, working with Ubuntu 16.4 (64bits) and GCC 5.4.0. We have verified that all what we proposed can pass all statistical tests of TestU01, from

SmallCruh to BigCrush. Let us recall that the permutation function [47] does not pass Crush and BigCrush when the space is lower than 36 bits, while in our case it does with only 32 bits and a lower modular multiplicative constant. Note that these results are naturally improved when we consider 64 outputted bits, leading to a better throughput.

D. Discussion

We have proven that if the generator provided in input is cryptographically secure, then the new generator resulting from our post-processing preserves this property. Of course, if the input generator is not safe (like the LFSR for example), ours is unlikely to be safe. In other words, this processing does not reduce the security of the generator provided in the input, and whether or not it is safe or not, this post-processing remains interesting. First of all, it has been proven that the output can change chaotically when the input is modified as long as the iteration graph is strongly connected. That is why, in connection with chaos, the output produced numbers can statistically appear as random (full success at TestU01), even if the input generator is statistically biased. In other words, the randomness is improved, at least from a statistical point of view. The resulting generator is fast, specifically designed for FPGA, and its architecture could eventually be massively parallelized on other architectures.

VII. ASIC IMPLEMENTATION ANALYSIS AND REAL-WORLD APPLICATIONS

In order to illustrate the hardware complexity of the PRNG based on generalized chaotic iterations, different ASIC implementation of our proposal are reported. Our GCIPRNG generators have been implemented and tested with TestU01 first with the previously defined permutation function (1) and next without such function but with generating a 32-bits number at each 5 cycles only. Indeed, an 65-*nm* CMOS Low leakage process technology node of UMC and Cadence flow $V14.2$ are used in our experiments. Table III summarizes the ASIC implementation, which uses two global flows: the synthesis *Cadence RTL Compiler* is based on the *Worst Case* library (WC=108°C and 1.08 Volt), while *Multi Mode Multi Corner* (matrix MMMC: 3−SDC × 3−Function-Mode × 2−Library) is applied for Place and Route flow (*Cadence Encounter Digital Implementation*) including both worst and best case library (BC=−40°C and 1.32 Volt). The obtained area in term of logic gate equivalent is the rate

$$\frac{area\ of\ P\&R\ (\mu m^2)}{area\ of\ logic\ AND(65nm)\ =\ 1.44\ (\mu m^2)}.$$

Additionally, the Static Timing Analysis (STA) and power analysis are obtained after *Signoff Verification* flow and *Gate-Level Simulation* (using *Switching Activity Interchange* (SAI)) for each MMMC. Meanwhile, some researchers do not consider the last two parameters for power estimation, which further induces a large difference when considering SAI information for dynamic power analysis.

Table III resumes the ASIC comparison results of GCIPRNG with other PRNGs for different applications and for each frequency band as: Low frequency 100Khz (LF), high frequency 13.56MHz (HF), and Ultra high frequency (UHF) (as defined in ISO/IEC 14223, 14443 [10], or 18000 [11]). Indeed, as noticed in the introduction, we consider the power and the throughput comparison for RFID EPC GEN-1&2 real applications under $100Khz$ ([18]). Our proposal GCIPRNG satisfies the power consumption constraints ($3.3\mu W$ to $7.8\mu W$) as it ranges between $5.4\mu W$ and $19\mu W$ [12] [13]. Moreover, the power consumption of our generators can be reduced if we consider a low power embedded strategy (linear PRNG) or disabling the permutation function. The same results for the area estimated less than 5000 GE as recommended [12]. Finally, even if the performance in terms of power consumption and the deployed area is lower than that of some other PRNG for these applications (RFID EPC GEN-1&2), our approach is the one that passes the whole TestU01 with the highest throughput. Additionally, based on HF and UHF frequencies, our generators provide the highest throughput (3.2Gbps to 9.4Gbps) and low power ($0.6mW$ to $2.3m$W) with less of 5% of total leakage power for different GCIPRNGs.

Finally, our generators GCIPRNG can be easily integrated to the internet of things (IoT,WSN) or smart card and is reconfigurable for any data length (8 to 128-bits). Moreover, our approach based on generating Boolean function by suppressing Hamilton cycle from generalized iteration graph equilibrium, can provide more than 1 million functions to be integrated. We can compare with an example of IoT [66] that uses MSP430g2553 (16-bits) requires approximately $25.8mA \Rightarrow 77.4mW$ (3V) and security part based on AES that consumes $24\mu W$ to generate 16-bits. Moreover, with the same frequency band (LF), our solution consumes between 3 to 7 times less energy.

VIII. CONCLUSION

In this research work, we have introduced a new chaotic PRNG implemented in FPGA, which is based on the combination of parallel executions of generalized chaotic iterations and of an efficient permutation scheme. Five Boolean functions have been iterated: the vectorial negation and four issued from removing a Hamilton cycle in the $\mathsf{N}-cube$. Three interesting strategy builders have been evaluated for each of them. These six variations lead to an hardware generator with one of the best throughput of the literature, and that can pass the most stringent statistical batteries of tests. If we consider the two conditions of throughput and statistics, we thus have obtained one of the best existing hardware generator. Last, but not least, we have rigorously proven too the chaotic and cryptographic behaviors of the whole proposal, and for the two Boolean functions.

REFERENCES

[1] L. D. Xu, W. He, and S. Li, “Internet of things in industries: A survey,” *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, Nov 2014.
[2] S. Li, T. Tryfonas, and H. Li, “The internet of things: a security point of view,” *Internet Research*, vol. 26, no. 2, pp. 337–359, 2016.
[3] Y. Zou and G. Wang, “Intercept behavior analysis of industrial wireless sensor networks in the presence of eavesdropping attack,” *IEEE Transactions on Industrial Informatics*, vol. 12, no. 2, pp. 780–787, April 2016.

[4] K. F. Tsang, M. Gidlund, and J. kerberg, "Guest editorial industrial wireless networks: Applications, challenges, and future directions," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 2, pp. 755–757, April 2016.

[5] M. Ma, D. He, N. Kumar, K. K. R. Choo, and J. Chen, "Certificateless searchable public key encryption scheme for industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 2, pp. 759–767, Feb 2018.

[6] Y. Liu, R. C. C. Cheung, and H. Wong, "A bias-bounded digital true random number generator architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 1, pp. 133–144, Jan 2017.

[7] X. Fang, B. Wetzel, J. M. Merolla, J. M. Dudley, L. Larger, C. Guyeux, and J. M. Bahi, "Noise and chaos contributions in fast random bit sequence generated from broadband optoelectronic entropy sources," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 3, pp. 888–901, March 2014.

[8] A. Vassilev and T. A. Hall, "The importance of entropy to information security," *Computer*, vol. 47, no. 2, pp. 78–81, Feb. 2014.

[9] A. Vassilev and R. Staples, "Entropy as a service: Unlocking cryptography's full potential," *Computer*, vol. 49, no. 9, pp. 98–102, Sept 2016.

[10] I. O. for Standardization, "Iso/iec 14443-1:2016 identification cards contactless integrated circuit cards proximity cards part 1: Physical characteristics," 2016-03.

[11] ——, "Information technology radio frequency identification for item management part 6: Parameters for air interface communications at 860 mhz to 960 mhz," 2010-12.

[12] J. Garcia-Alfaro, M. Barbeau, and E. Kranakis, "Security threat mitigation trends in low-cost rfid systems," in *Data Privacy Management and Autonomous Spontaneous Security*, J. Garcia-Alfaro, G. Navarro-Arribas, N. Cuppens-Boulahia, and Y. Roudier, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 193–207.

[13] M. Feldhofer and J. Wolkerstorfer, *Hardware Implementation of Symmetric Algorithms for RFID Security*. Boston, MA: Springer US, 2008, pp. 373–415.

[14] D. Seetharam and S. Rhee, "An efficient pseudo random number generator for low-power sensor networks [wireless networks]," in *29th Annual IEEE International Conference on Local Computer Networks*, Nov 2004, pp. 560–562.

[15] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. E. Tapiador, E. S. Millán, and J. C. A. van der Lubbe, "Security flaws in an efficient pseudo-random number generator for low-power environments," in *Security in Emerging Wireless Communication and Networking Systems*, Q. Gu, W. Zang, and M. Yu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 25–35.

[16] A. Francillon and C. Castelluccia, "Tinyrng: A cryptographic random number generator for wireless sensors network nodes," in *2007 5th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks and Workshops*, April 2007, pp. 1–7.

[17] K. Mandal, X. Fan, and G. Gong, "Design and implementation of warbler family of lightweight pseudorandom number generators for smart devices," *ACM Trans. Embed. Comput. Syst.*, vol. 15, no. 1, pp. 1:1–1:28, Feb. 2016.

[18] E. S. Specification, "Epc radio-frequency identification protocols generation-2 uhf rfid: Specifications for rfid air interface protocols for communications at 860 mhz-960 mhz," Apr-2015. [Online]. Available: https://www.gs1.org/sites/default/files/docs/epc/Gen2_Protocol_Standard.pdf

[19] M. Joseph, G. Sekar, and R. Balasubramanian, "Distinguishing attacks on (ultra-)lightweight wg ciphers," in *Lightweight Cryptography for Security and Privacy*, A. Bogdanov, Ed. Cham: Springer International Publishing, 2017, pp. 45–59.

[20] A. B. Orúe, L. Hernández Encinas, V. Fernández, and F. Montoya, "A review of cryptographically secure prngs in constrained devices for the iot," in *International Joint Conference SOCO'17-CISIS'17-ICEUTE'17 León, Spain, September 6–8, 2017, Proceeding*, H. Pérez García, J. Alfonso-Cendón, L. Sánchez González, H. Quintián, and E. Corchado, Eds. Cham: Springer International Publishing, 2018, pp. 672–682.

[21] S. Contassot-Vivier, J.-F. Couchot, C. Guyeux, and P.-C. Heam, "Random walk in a n-cube without hamiltonian cycle to chaotic pseudo-random number generation: Theoretical and practical considerations," *International Journal of Bifurcation and Chaos*, vol. 27, no. 01, p. 1750014, 2017.

[22] M. Bakiri, J.-F. Couchot, and C. Guyeux, "One random jump and one permutation: Sufficient conditions to chaotic, statistically faultless, and large throughput prng for fpga," in *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications - Volume 6: SECRYPT, (ICETE 2017)*, INSTICC. SciTePress, 2017, pp. 295–302.

[23] P. L'Ecuyer and R. Simard, "Testu01: A c library for empirical testing of random number generators," *ACM Transactions on Mathematical Software (TOMS)*, vol. 33, no. 4, pp. 22:1–22:40, Aug. 2007. [Online]. Available: http://doi.acm.org/10.1145/1268776.1268777

[24] P. Dabal and R. Pelka, "A chaos-based pseudo-random bit generator implemented in fpga device," in *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, April 2011, pp. 151–154.

[25] ——, "Fpga implementation of chaotic pseudo-random bit generators," in *Proceedings of the 19th International Conference Mixed Design of Integrated Circuits and Systems - MIXDES 2012*, May 2012, pp. 260–264.

[26] M. Hénon, "A two-dimensional mapping with a strange attractor," *Communications in Mathematical Physics*, vol. 50, no. 1, pp. 69–77, Feb 1976. [Online]. Available: https://doi.org/10.1007/BF01608556

[27] P. Dabal and R. Pelka, "A study on fast pipelined pseudo-random number generator based on chaotic logistic map," in *17th International Symposium on Design and Diagnostics of Electronic Circuits Systems*, April 2014, pp. 195–200.

[28] A. Pande and J. Zambreno, "Design and hardware implementation of a chaotic encryption scheme for real-time embedded systems," in *2010 International Conference on Signal Processing and Communications (SPCOM)*, July 2010, pp. 1–5.

[29] S. Liu, J. Sun, Z. Xu, and Z. Cai, "An improved chaos-based stream cipher algorithm and its vlsi implementation," in *2008 Fourth International Conference on Networked Computing and Advanced Information Management*, vol. 2, Sept 2008, pp. 191–197.

[30] P. Giard, G. Kaddoum, F. Gagnon, and C. Thibeault, "Fpga implementation and evaluation of discrete-time chaotic generators circuits," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, Oct 2012, pp. 3221–3224.

[31] J. Černák, "Digital generators of chaos," *Physics Letters A*, vol. 214, no. 3, pp. 151 – 160, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/037596019600179X

[32] C.-Y. Li, J.-S. Chen, and T.-Y. Chang, "A chaos-based pseudo random number generator using timing-based reseeding method," in *2006 IEEE International Symposium on Circuits and Systems*, May 2006, pp. 4 pp.–3280.

[33] C. Y. Li, Y. H. Chen, T. Y. Chang, L. Y. Deng, and K. To, "Period extension and randomness enhancement using high-throughput reseeding-mixing prng," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 2, pp. 385–389, Feb 2012.

[34] M. A. Zidan, A. G. Radwan, and K. N. Salama, "The effect of numerical techniques on differential equation based chaotic generators," in *ICM 2011 Proceeding*, Dec 2011, pp. 1–4.

[35] ——, "Random number generation based on digital differential chaos," in *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2011, pp. 1–4.

[36] H. Hu, L. Liu, and N. Ding, "Pseudorandom sequence generator based on the chen chaotic system," *Computer Physics Communications*, vol. 184, no. 3, pp. 765–768, 2013.

[37] M. Bakiri, J.-F. Couchot, and C. Guyeux, "Fpga implementation of f2-linear pseudorandom number generators based on zynq mpsoc: A chaotic iterations post processing case study," in *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications - Volume 4: SECRYPT,*, 2016, pp. 302–309.

[38] Q. Wang, S. Yu, C. Li, J. L, X. Fang, C. Guyeux, and J. M. Bahi, "Theoretical design and fpga-based implementation of higher-dimensional digital chaotic systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 3, pp. 401–412, March 2016.

[39] NIST. (2010) National institute of standards and technology (nist): A statistical test suite for random and pseudorandom number generators for cryptographic applications @ONLINE. [Online]. Available: csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf

[40] M. Bakiri, C. Guyeux, J.-F. Couchot, and A. K. Oudjida, "Survey on hardware implementation of random number generators on fpga: Theory and experimental analyses," *Computer Science Review*, vol. 27, pp. 135 – 153, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013716302271

[41] M. Bakiri, J. F. Couchot, and C. Guyeux, "Ciprng: A vlsi family of chaotic iterations post-processings for $mathbbF_2$ -linear pseudorandom number generation based on zynq mpsoc," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 5, pp. 1628–1641, May 2018.

[42] J. Bahi and C. Guyeux, "An improved watermarking algorithm for Internet applications," in *INTERNET'2010. The 2nd Int. Conf. on*

Evolving Internet, Spain, Jan. 2010, pp. 119 – 124. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00956825

[43] J.-F. Couchot, P.-C. Heam, C. Guyeux, Q. Wang, and J. M. Bahi, "Pseudorandom number generators with balanced gray codes," in *Security and Cryptography (SECRYPT), 2014 11th International Conference on*. IEEE, 2014, pp. 1–7.

[44] J. M. Bahi, X. Fang, C. Guyeux, and L. Larger, "Fpga design for pseudorandom number generator based on chaotic iteration used in information hiding application," *Appl. Math*, vol. 7, no. 6, pp. 2175–2188, 2013.

[45] X. Fang, Q. Wang, C. Guyeux, and J. M. Bahi, "Fpga acceleration of a pseudorandom number generator based on chaotic iterations," *Journal of Information Security and Applications*, vol. 19, no. 1, pp. 78–87, 2014.

[46] J. M. Bahi, R. Couturier, C. Guyeux, and P.-C. Heam, "Efficient and cryptographically secure generation of chaotic pseudorandom numbers on gpu," *The journal of Supercomputing*, vol. 71, no. 10, pp. 3877–3903, oct 2015. [Online]. Available: http://arxiv.org/pdf/1112.5239.pdf

[47] M. E. O'Neill, "Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation," Harvey Mudd College, Claremont, CA, Tech. Rep. HMC-CS-2014-0905, Sep. 2014.

[48] S. Contassot-Vivier and J. Couchot, "Canonical form of gray codes in n-cubes," in *Cellular Automata and Discrete Complex Systems - 23rd IFIP WG 1.5 International Workshop, AUTOMATA 2017, Milan, Italy, June 7-9, 2017, Proceedings*, ser. Lecture Notes in Computer Science, A. Dennunzio, E. Formenti, L. Manzoni, and A. E. Porreca, Eds., vol. 10248. Springer, 2017, pp. 68–80.

[49] C. Guyeux and J. M. Bahi, *A Topological Study of Chaotic Iterations Application to Hash Functions*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 51–73. [Online]. Available: https://doi.org/10.1007/978-3-642-25237-2_5

[50] X. Fang, B. Wetzel, J.-M. Merolla, J. M. Dudley, L. Larger, C. Guyeux, and J. M. Bahi, "Noise and chaos contributions in fast random bit sequence generated from broadband optoelectronic entropy sources," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 3, March 2014.

[51] C. Guyeux, *Le désordre des itérations chaotiques - Applications aux réseaux de capteurs, à la dissimulation d'information, et aux fonctions de hachage*. Éditions Universitaires Européennes, 2012, iSBN 978-3-8417-9417-8. 362 pages. Publication de la thse de doctorat.

[52] R. L. Devaney, *An Introduction to Chaotic Dynamical Systems, 2nd Edition*. Westview press, March 2003.

[53] J. Banks, J. Brooks, G. Cairns, G. Davis, and P. Stacey, "On devaney's definition of chaos," *The American Mathematical Monthly*, vol. 99, no. 4, pp. 332–334, 1992.

[54] Y. Deng, H. Hu, N. Xiong, W. Xiong, and L. Liu, "A general hybrid model for chaos robust synchronization and degradation reduction," *Information Sciences*, vol. 305, no. Complete, pp. 146–164, 2015.

[55] G. Yang, M. D. Aagaard, and G. Gong, "Efficient hardware implementations of the warbler pseudorandom number generator." *IACR Cryptology ePrint Archive*, vol. 2015, p. 789, 2015.

[56] K. Mandal, X. Fan, and G. Gong, "Design and implementation of warbler family of lightweight pseudorandom number generators for smart devices," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 1, p. 1, 2016.

[57] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. Estevez-Tapiador, and A. Ribagorda, "Lameda prng for epc class-1 generation-2 rfid specification," *Computer Standards & Interfaces*, vol. 31, no. 1, pp. 88–97, 2009.

[58] J. Melia-Segui, J. Garcia-Alfaro, and J. Herrera-Joancomarti, "Analysis and improvement of a pseudorandom number generator for epc gen2 tags," in *Financial Cryptography and Data Security*, R. Sion, R. Curtmola, S. Dietrich, A. Kiayias, J. M. Miret, K. Sako, and F. Sebé, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 34–46.

[59] J. Meli-Segu, J. Garcia-Alfaro, and J. Herrera-Joancomart, "J3gen: A prng for low-cost passive rfid," *Sensors*, vol. 13, no. 3, pp. 3816–3830, 2013.

[60] H. Martin, E. San Millán, P. Peris-Lopez, and J. E. Tapiador, "Efficient asic implementation and analysis of two epc-c1g2 rfid authentication protocols," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3537–3547, 2013.

[61] M. D. Aagaard, G. Gong, and R. K. Mota, "Hardware implementations of the wg-5 cipher for passive rfid tags," in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, June 2013, pp. 29–34.

[62] G. Yang, X. Fan, M. Aagaard, and G. Gong, "Design space exploration of the lightweight stream cipher wg-8 for fpgas and asics," in *Proceedings of the Workshop on Embedded Systems Security*. ACM, 2013, p. 8.

[63] S. K. Mathew, S. Srinivasan, M. A. Anders, H. Kaul, S. K. Hsu, F. Sheikh, A. Agarwal, S. Satpathy, and R. K. Krishnamurthy, "2.4 gbps, 7 mw all-digital pvt-variation tolerant true random number generator for 45 nm cmos high-performance microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 11, pp. 2807–2821, Nov 2012.

[64] M. Bucci, L. Germani, R. Luzzi, A. Trifiletti, and M. Varanonuovo, "A high-speed oscillator-based truly random number source for cryptographic applications on a smart card ic," *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 403–409, April 2003.

[65] P. Giard, G. Kaddoum, F. Gagnon, and C. Thibeault, "Fpga implementation and evaluation of discrete-time chaotic generators circuits," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, Oct 2012, pp. 3221–3224.

[66] W. Trappe, R. Howard, and R. S. Moore, "Low-energy security: Limits and opportunities in the internet of things," *IEEE Security Privacy*, vol. 13, no. 1, pp. 14–21, Jan 2015.

Mohammed Bakiri is a research engineer in the department of Microelectronique at "Centre de Développement des Technologie Avancées", CDTA, Algiers, in 2010. He received a Ph.D. in Computer Science in 2018 in the FEMTO-ST institute, Bourgogne Franche-Comté University, France. He is presently working on new chaotic algorithms for lightweight cryptography and random number generators dedicated to FPGA/ASIC.

Christophe guyeux The research interests of Pr. Guyeux are in the areas of interdisciplinary sciences and complex systems. He applies techniques from mathematics and/or computer science to solve scientific questions raised in biology, environment, or computer science fields. Current areas of application in computer science include chaos study of discrete dynamical systems applied to information security and wireless sensor networks. In multidisciplinary sciences, he contributes to solve issues raised by colleagues in biology and environment.

Jean-François Couchot is an Associate Professor in the Department of Computer Science (DISC) of the FEMTO-ST institute, Bourgogne Franche-Comté University. He received a Ph.D. in Computer Science in 2006 in the FEMTO-ST institute. He has applied for a postdoctoral position at INRIA Saclay Ile de France in 2006. His research focuses on discrete dynamic systems (data hiding, pseudorandom number generators, hash function) and on bioinformatics, especially in gene evolution prediction. He has written more than 25 scientific articles in these areas.

Luigi Marangio received his Master degree in Mathematics from the University of Pisa in 2017 and he is currently Ph.D student in Computer Science in the Femto-ST Institute, Bourgogne Franche-Comté University, France. His research project involves the study of dynamical systems in several situations, with particular focus on pseudorandom number generators and some not well-understood chaotic phenomena that comes out from different situations in science (for instance, toy models in biology or chemistry).

Stefano Galatolo is Associate Professor at Dipartimento di Matematica, University of Pisa and director of Centro Interdipartimentale per lo Studio dei Sistemi Complessi. His research focuses on dynamical systems, its statistical behavior and computational methods. He is author of about 50 papers in these fields and currently editor of Chaos Solitons and Fractals and Journal of Fixed Point Theory and Applications. .