Martin, H., Di Natale, G. & Entrena, L. (2018). Towards a Dependable True Random Number Generator With Self-Repair Capabilities. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(1), pp. 247–256.

DOI: 10.1109/tcsi.2017.2711033

Towards a dependable True Random Number Generator with self-repair capabilities

Honorio Martin, Giorgio Di Natale and Luis Entrena

Abstract—**Many secure-critical systems rely on true random number generators that must guarantee their operational functionality during its intended life. To this end, these generators are subject to intensive on-line testing in order to discover any flaws in their operation. The dependability of the different blocks that compose the system is crucial to guarantee the security. In this work we provide some general guidelines for designers to create more dependable TRNGs. In addition, a case of study where the system dependability has been improved is presented.**

Index Terms—**TRNGs, dependability, self-repairable, on-line testing**

I. Introduction

Nowadays, for many electronic and telecommunication systems, one of the most important system properties is dependability. Among other aspects, dependability includes the ability of the system to protect itself against accidental or deliberate intrusions [5]. Generally, attackers will focus their efforts against security blocks in order to get access to sensitive information. So, it is important to implement dependable cryptographic protocols in order to assure the security of the system.

The security of many cryptographic systems is related to the implementation of secure cryptographic algorithms and communication protocols requiring the use of random numbers. These numbers are typically generated inside the system by a True Random Number Generator (TRNG). Among the requirements of the generated random numbers stand out good statistical properties and unpredictability. Many TRNGs that fulfil these requirements have been proposed in the scientific literature [33][30][34]. In parallel to the development of TRNGs, there is an increasing demand for other blocks associated to the random generation as tests that guarantee TRNG randomness in different scenarios or post-processing blocks that correct the bias at the output of a TRNG.

These tests usually check for the statistical quality of the output and the entropy source. Among the first ones stand out the NIST suite [7] and the AIS-31 tests [4]. These tests can be divided into those that check for the statistical quality of the raw data [3] and those that test some physical parameter that is closely related to the entropy of the source of randomness[31].

Nevertheless, there is an open question in this topic: what has to be done with a particular generator when a test fails? To the best of our knowledge, all the aforementioned TRNG solutions raise an alarm but they provide no specification about appropriate actions to recover in case of a fault. These designs are therefore able to obtain high-entropy TRNGs that, nevertheless, will be out of service in case of fault.

In addition, on-the-fly tests and post-processing blocks can be the source of faults in a TRNG. The deactivation of the alarm test while the post-processing is attacked can lead to catastrophic consequences in the key generation process. Even just a controlled bias due to the malfunctioning of the post-processing block can improve the chances for a subsequent cryptanalysis. All in all, a realistic scenario where the embedded tests and post-processing blocks will be the target of attacks must be considered.

In this work we attempt to provide some general guidelines for designers to create more dependable TRNGs. These guidelines will take into account special considerations related to the indeterministic nature of TRNGs and possible scenarios where on-the-fly tests and post-processing are faulty. In addition, we have proposed a novel architecture that is able to operate and self-recover from different fault scenarios. Our architecture includes support for some of the aforementioned guidelines and redundant blocks that allow us to detect, locate and counteract faults in all the blocks involved in the random generation. Through a case of study, we have illustrated and demonstrated the capabilities of the proposed architecture. The dependability of this proposal has been evaluated by simulating the injection of different kinds of faults.

The remainder of this paper is organized as follows. Section II presents some guidelines to create dependable TRNGs. Section III describes a case of study where the aforementioned guidelines are applied. Section IV reports the impact of the aforementioned guidelines in the main metrics (logic resources, power consumption and throughput). In Section IV-B, we validate the system dependability by the simulation of different faults. Section V summarizes the conclusions of this work.

II. General guidelines for dependable TRNGs

TRNGs play a key role in many security systems. They are typically dedicated to generate keys, nonces, padding plaintext, etc. As their correct operation is crucial, special care must be taken to ensure that the TRNG is dependable against the multiple threats they are subject to.

Dependability is defined as the ability of a system to deliver its intended level of service to its users [16]. Regarding TRNGs, the term dependability is closely related to the guarantee of a minimum entropy and a good statistical distribution of the generated data. Therefore, TRNG dependability will

H. Martin and L. Entrena are with Department of Electronics Technology, Universidad Carlos III de Madrid,(e-mail: (hmartin,entrena)@ing.uc3m.es)

G. Di Natale is with the LIRMM, Universite Montpellier II,(e-mail: DiNatale@lirmm.fr)

depend on the TRNG blocks that generate and guarantee the aforementioned features.

Fig. 1 shows the main blocks of a TRNG. A typical TRNG is composed of a digitized noise source, from where the entropy is extracted, a post-processing block that corrects the bias and finally some on-the-fly tests that check for the statistical quality of the generated data. As typical TRNGs are composed of some digital blocks, widespread hardening solutions that are applied to general-purpose digital circuits can be considered to increase TRNG dependability. Nevertheless, the indeterministic nature of TRNGs makes it necessary to modify somehow these well known techniques and integrate some new ones.

It is important to note that the applied strategies will depend on the target application (and its corresponding security levels), the specific TRNG (e.g., entropy extraction mechanism, post-processing technique, etc.) and the available resources. In the following subsections we present a number of recommendations that are valid to increase TRNG dependability. We can group them all into three distinct phases: prevention, detection, and response.

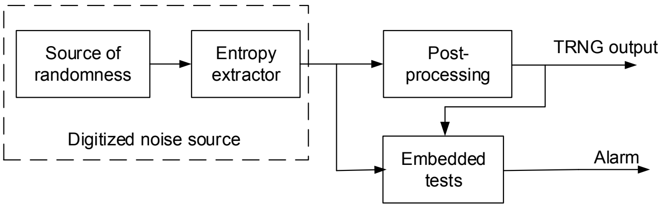


Fig. 1. TRNG general scheme

1) Prevention: In this group, methods and techniques focused on fault prevention, fault tolerance, and fault forecasting are included. Among the different techniques stand out:

- **Reliable Finite State Machines (FSMs):** FSMs are important parts of TRNGs. They are used not only for the flow control but also in the typical on-the-fly embedded tests. The well known one-hot encoding with error detection (parity checker) can be used to increase the system dependability with a little penalty in area. Other solutions as duplication with comparison or triple modular redundancy (TMR) are eligible to enhance FSMs dependability [10].
- **Control surrounding conditions:** It is known that fluctuations on environmental or operational conditions can cause a bias in the TRNG output [27]. It could be interesting to check some of the main magnitudes that affect randomness (Temperature, Voltage, etc.) in order to foresee and prevent a faulty behaviour. Early-warning threat detection methods, as the canary numbers presented in [32], are an interesting option. For FPGA implementations, real time monitoring systems that check the on-chip sensors can be considered as a straightforward solution (e.g. Xilinx IP XADC [20]).
- **Attack countermeasures:** TRNGs are susceptible to many different attacks because of their paramount importance in critical systems. In recent times, several attacks have been presented against some specific TRNGs [29][14][24]. Some of these attacks can be thwarted by adding lightweight countermeasures in the original design. For example, temperature or underpowering attacks can be tackled by reducing the critical path delay of the circuit. Another solution could be the integration of a clock-observation circuit in order to ensure a glitch-free clock signal [14].
- **Anti-Aging effects:** Circuit aging refers to the deterioration of circuit performance over time. In the specific case of a TRNG, aging can evolve in an unacceptable degradation on the randomness. It is known that one of the main reasons for aging is the switching activity. SRAM or Ring Oscillator (RO) TRNGs that constantly change states are more susceptible to aging. Clock gating or enable gating techniques that *disconnect* some blocks of the circuit can be used as a solution for RO TRNGs. For SRAM TRNGs, the overall lowest aging is achieved when both PMOS transistors are stressed equally during the whole lifetime [18].

2) Detection: In order to accomplish the self-repairability of a TRNG, it is a key aspect to be able to detect different kinds of faults. Typical scenarios considered in the scientific literature only include the evaluation of faults in the digitized noise source (source of randomness and entropy extractor). Post-processing blocks and on-the-fly tests must be included as a possible source of faults. Taking this new scenario into account, it is crucial that the detection strategies answer the following questions: Where is the fault located? How bad/harmful is the fault?

Regarding the fault location, designers can make use of embedded tests in order to identify the faulty component. These embedded tests are typically designed to check the minimun entropy generated in the noise source [3] or the statistical quality of the final output [7]. Consequently, they can detect a fault in the noise source or in the post-processing. Some interesting lightweight ad-hoc solutions, that take into account the TRNG stochastic model, could allow the detection of faults on the digitized noise source [31]. Other interesting approximation based on the effects of typical attacks performed against TRNGs was presented in [9]. On the other hand, redundancy might be applied to detect faults in the on-board supported tests.

Assessing the harmfulness of a fault is also important in order to take effective countermeasures for recovery. In the TRNG case, the harmfulness can be evaluated by using the embedded tests. For tests that check the noise source, the most extended approximation relies on the estimation of minimun entropy [3]. Depending on the application, different ranges of minimum entropy could be established in order to distinguish between an acceptable degradation and a fault that compromises the randomness. In the same way, it is possible to modify the statistical tests of the output in order to establish several ranges to differentiate different scenarios. An appropriate assessing of the fault can avoid an unnecessary TRNG shut-down.

3) Response: Once a fault has been detected, the first step is to consider if the TRNG must be stopped. In the vast majority of applications, a TRNG must not work if a certain level of entropy or a good statistical distribution is not guaranteed at the output. Consequently, the TRNG must not work if the embedded tests fail. In all other cases, stopping the TRNG will highly depend on the application. The response to a fault should be planned well in advance. Once this first decision is made, additional actions focused on regaining the TRNG operational status should be taken. These actions will depend on the available resources, the application and the TRNG itself. For example, if our application can tolerate certain degradation on the minimum entropy, an on-line enhancement of the post-processing could counteract the fault effects. In the scientific literature, some ad-hoc solutions that can be used for very specific scenarios have been proposed [19][12]. More specifically, in [19] the authors proposed a bias detector that control two tunable ROs to avoid interlocking between them. In [12], authors increase the compression factor or decrease the sampling frequency of a RO-TRNG depending on the distribution of the output.

The aforementioned techniques have been applied in isolation trying to enhance a specific block (typically the digitized noise source). However, it is important to remark that the aforementioned techniques must work all together in order to reach the system dependability. Moreover, a selection of the different techniques must be integrated in an efficient manner in order to offer effective countermeasures. All in all, once the operational framework has been established (application, TRNG, resources, attack scenarios, etc.), the designer needs to select different prevention, detection and response techniques that ensure the system functionality during its intended operation life.

In this work, we propose the integration of several of these techniques within a novel architecture. Key novelties in this architecture are the duplication of noise sources, entropy tests and post-processing modules, the use of a LFSR for both testing and operation modes and a checker+controller module that can guarantee dependability even in the presence of faults and even under some attacks. As a result, our approach is able to continue operation with acceptable performance in many situations where previous techniques would stop operation or simply fail. To the best of our knowledge, this is the first work where the entire system dependability is addressed and demonstrated.

III. A CASE OF STUDY

In this section we present the proposed architecture and a case of study where we have added support for some of the aforesaid guidelines (redundancy, one-hot coding, control main magnitudes, etc.). This architecture is flexible enough to adapt to different requirements by using a variety of modules. Through the case of study we illustrate and demonstrate its capabilities. We have defined a scenario where a RO-TRNG is used to generate nonces and padding plain-texts. Certain degradation on the entropy source is allowed. In addition, no limits in terms of resources are established. The main goals of our design are to always guarantee a minimum entropy level and to offer self-recovery capabilities from different scenarios.

A. Fault attack scenarios and fault models

Fault attacks have been widely studied with the purpose of either extracting sensitive information from a secure device, or obtaining denial-of-service. In TRNGs, fault attacks can be used to bias the randomness of the random source so that predictable numbers can be obtained. While an adversary can use any kind of fault for performing an attack with catastrophic consequences, only a subset of those attacks can be actually exploited in order to retrieve useful information. In order to limit the study to realistic and exploitable attacks, we have considered the following fault models and scenarios:

- **Bit-flip:** A bit flip fault consists in a flipping of a single bit value stored in a single memory cell. The bit flip can affect any bit at any time during the execution. Typically, the bit flip results from a premature power drain on one of the register cells that can be carried out using underfeeding or power spikes [11]. Induced electrical fields and Eddy currents attacks can also lead to this kind of faults [25]. In high-reliability FPGA implementations, single event upset (SEU) caused by a ionizing particles are a growing concern.
- **Stuck-at:** A stuck-at fault transforms the correct value of a line to a constant logic value, either a logic 0 or a logic 1. The effects of stuck-at faults are typically considered as permanent. In practice, the stuck-at model covers many of the possible manufacturing defects in CMOS circuits or destructive faults, where it is assumed that a destroyed wire, gate or memory cell will cause the faulty bit to be constant. A laser cutter or UV light can be used for destroying individual structures of the chip that can derive in stuck-at faults [22] [13].
- **Operating condition variations:** Temperature and power supply voltage variations can induce faults through timing violations. In particular, the increasing of the temperature or underpowering will increase the critical path delay causing a malfunctioning [28]. Several temperature and underpowering attacks have been reported in the literature [27][14]. A bias at the output is the typical outcome of these attacks in TRNGs.

B. TRNG blocks

Fig. 2 shows a block diagram of our proposal. For the sake of simplicity, some signals, blocks and multiplexers have been omitted. Redundant blocks have been noted as A and B. The basic idea of our proposed architecture is the following: the noise sources are checked by 2 entropy test blocks that work in parallel. The raw output of the noise source (A or B) is post-processed by either a post-processing block (A or B) or the LFSR. Then, the output is tested by two FIPS-140 blocks working in parallel. Finally, the outputs of the different tests and the surrounding conditions are constantly evaluated by the checker and controller block. The redundant blocks and the different modifications of our scheme give the system sufficient flexibility to work even under the aforementioned attack scenarios.

The different block modifications to counteract the aforementioned scenarios are explained in detail in the following paragraphs.

1) Noise Source: **(Inputs: Enable, RST; Outputs: NS-X). We have selected the RO based TRNG presented by Sunar et al. in [2] because of its straightforward implementation and flexibility. In this TRNG the outputs of several ring oscillators are sampled and then collected through an XOR-tree to obtain a single bit at the output. The principle of this generator is simple: if at least one of the high frequency signals is sampled on the jitter zone, the output will be random. It seems intuitive that increasing the number of high frequency signals implies that it is more likely to sample one in the jitter zone. The main drawback of this design is that the XOR gate cannot properly handle the high number of transitions that are produced by the ROs. In order to overcome this flaw, Wold et al.[15] proposed a modified version that includes a flip-flop at the output of each RO (before the XOR gate). The general scheme of this modified TRNG is depicted in Fig. 3. As stated in [21], the mathematical proof presented by Sunar et al. in the original work remains valid after the modification. We have designed our enhanced TRNG taking into consideration the original stochastic model. The number of ROs have been obtained using expresion (1)[23].**

$$m >= \frac{T}{\sigma_{acc}} \tag{1}$$

where m is the number of ROs, T is the clock period and σ_{acc} the accumulated jitter during a period. In order to avoid a higher number of RO, we have decided to accumulate entropy during 100 clock cycles. Taking into account the measured accumulated jitter ($\sigma_{acc} = 33ps$) and the selected sampling frequency of 300 MHz, the number of ROs that are necessary to fulfil the stochastic model are 102 ROs The value obtained is similar to that published in the original work (N=114)[2] and in [23]. However, the final number of ROs used in our proposal has been incremented by 25% (128 ROs) in order to tackle attacks that take advantage of the RO locking [21]. Finally, our enhanced TRNG consists of two redundant noise sources (NS-A and NS-B) that will be used alternatively depending on the scenario. Each noise source has been oversized in order to handle different scenarios. More specifically, each noise source is composed of 256 ROs where only 128 ROs are active in normal conditions. As shown in Fig. 4, each oscillator consists of 4 inverters and one NAND gate, the latter being able to enable/disable the oscillation. The XOR-tree (see Fig. 5) uses a ripple structure in order to avoid the effects of power and clock glitches as those presented in [14].

***2) Post-processing*:** **(Inputs: RST, NS-X, select_order; Outputs: PP-X).** We have selected an n^{th} order parity filter as a post-processing because of its straightforward implementation. More precisely, using a shift register and XOR gates, the parity filter hashes n successive input bits into one output bit (Fig. 6). This post-processing enhances the entropy per output bit, but reduces the throughput n times. Taking into account the stochastic model introduced in the previous section, we have selected a 100-th order filter in normal conditions and we will increase the jitter accumulation time by a 10% depending on the scenarios. We have modified the parity filter in order to be able to select on-the-fly the filter order (100, 110, 120 and 130-th order). PP-A and PP-B blocks of Fig. 2 correspond to these redundant post-processing blocks.

3) Embedded Tests: The embedded tests in our proposal check two different parameters: the minimum entropy generated by the noise source and the statistical distribution of the final output. All the tests have been duplicated in order to detect faults on themselves.

a) Entropy Tests: **(Inputs: RST, NS-X, LFSR; Outputs: Low_entropy_Alarm, Medium_entropy_alarm, Counters).** As there is no claim about the noise source to be identically distributed, we have selected three NIST tests that are oriented to estimate the mininimun entropy of non identically distributed (non-IID) number generators [3]. These generators may have dependencies in time and/or state that may result in an overestimation of entropy if typical test suites are used.

The three entropy tests have been selected attending to their straightforward implementation. We have adopted some working assumptions that simplify the implementation of these tests [8]. A random bit will be generated each clock cycle, which implies that the output space is two (0 or 1). The dataset length will be 2^{13}. Cut-off values have been precomputed for each test in order to determine different entropy levels (low, medium and high entropy). Two alarms will be generated: *medium entropy alarm* and *low entropy alarm*. A brief description of each test is presented below.

The Frequency Test: This test computes the number of occurrences of the most-likely sample value. The probability distribution of the samples is modelled from a noise source. More accurate results can be obtained with a large dataset. A compact implementation of this test can be obtained by using an up/down counter.

The Collision Test: A collision can be described as a subsequence of repeated values in a dataset. In our case, the TRNG only generates two different values (0 or 1), thus a collision will take place after 2 bits (if they are equal) or after 3 bits (if the first 2 bits are different). The aim of this test is to determine the mean collision time until the end of the dataset is reached by tracking the number of 2 and 3 bits sub-sequences that contain a collision. [8]. A lightweight implementation can be obtained using a simple FSM to detect 3-bits segments that contain a collision and generate the enable of a counter.

The Partial Collection Test: This test computes the entropy of a dataset based on how many distinct values in the output space are observed. To that end, the dataset is divided into non-overlapping subsets of n bits, where n is the size of the output space. In this case, it is observed the number of times that two different vectors {01,10} appear. A two bit shift register and a XOR gate connected to the enable of a counter are sufficient to implement this test.

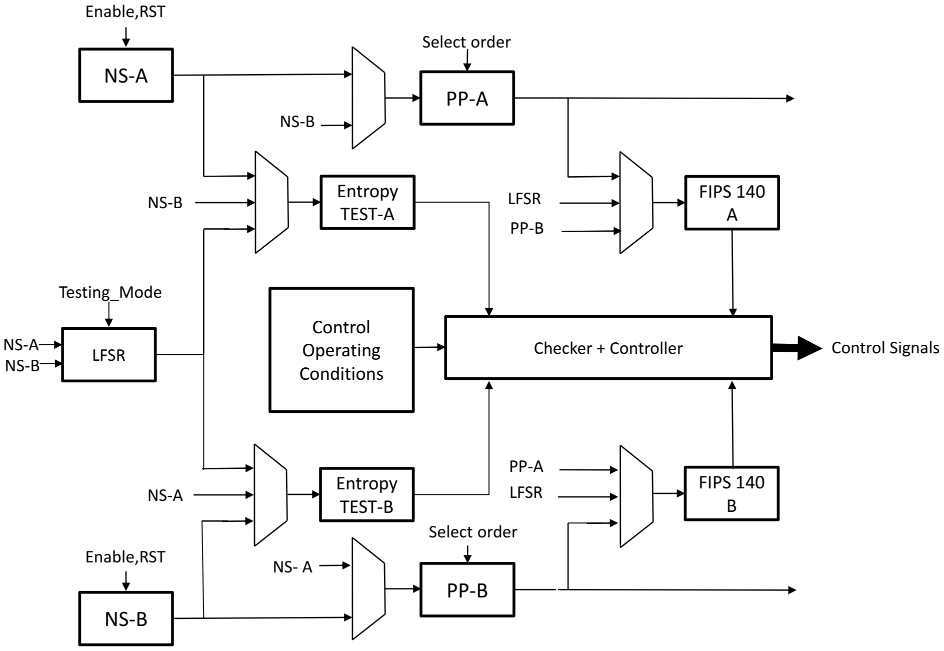


Fig. 2. Self-repairable TRNG scheme

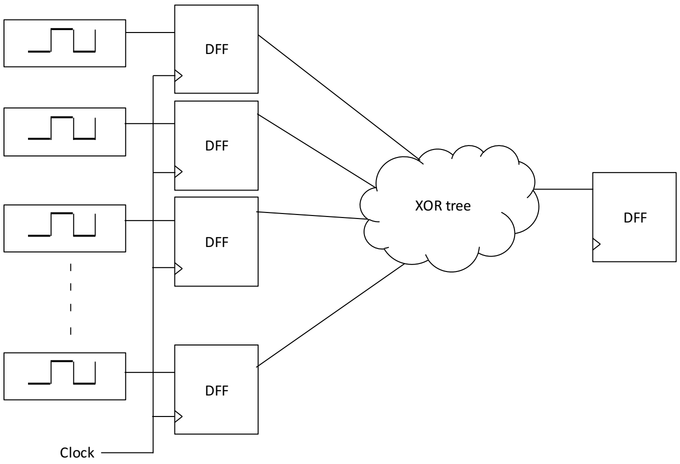


Fig. 3. Sunar et al. modified scheme

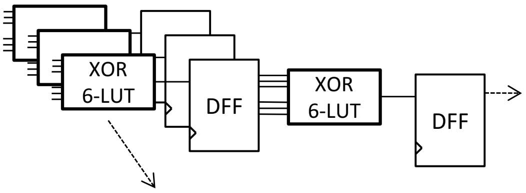


Fig. 5. Implemented XOR-tree ripple structure

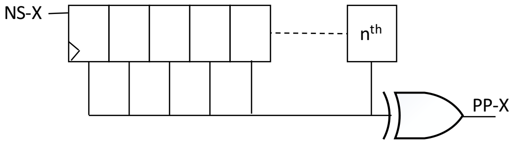


Fig. 6. n^{th} order parity filter

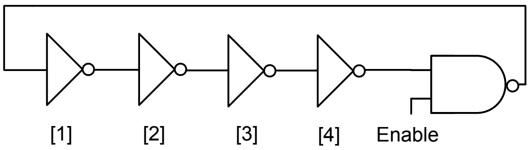


Fig. 4. RO with enable

b) Statistical output Tests: **(Inputs: RST, PP-X, LFSR; Outputs: FIPS_Alarm, Counters).** We adopted the Federal Information Processing Standard (FIPS) 140 [1] among published statistical tests, which consists of the following tests: Monobit, Poker, Runs, and Long run. The required length of the sequence is 20000 bits, which implies small demand on hardware resources. We have precomputed the cut-off values for each test that gives us an output (pass/no pass).

Monobit: This test is analogous to the aforementioned frequency test. In this case, it counts the number of ones. Only a counter is necessary to implement this test.
Poker: The dataset is divided in 4-bit segments and the number of occurrences of each of the 16 possible 4-bit values is counted. We have used the simplified equation presented in [17] to carry out the final computation required in the original test ($1563175 < \sum_{i=0}^{1} 5f(i)^2 < 1576929$). A FSM and 16 counters are necessary to find the number of occurrences of each 4-bit value. A DSP block is used to calculate a square in the final computation.
Runs: A run is defined as a maximal sequence of consecutive bits of either all ones or all zeros. The test is passed if the number of runs is within the corresponding

interval specified [17]. An FSM and several counters are needed to implement this test.

Long run: A long run is defined as a run of length 26 or more (of either zeros or ones)[1]. The test is passed if there are no long runs. This test can be implemented using a counter and simple logic.

It is important to remark that we have used one-hot encoding with self-recovery for all the FSMs involved in the different tests. In addition, some output counters were used for testing purposes. We have estimated a min-entropy of 0.9942 for normal TRNG operation.

4) LFSR: **(Inputs: RST, NS-A, NS-B, Testing_Mode; Outputs: LFSR).** Fig. 7 shows a modified Galois LFSR that is included in our enhanced design. We have implemented a 64-bit LFSR using a primitive polynomial to reach maximal LFSR length. This LFSR has two operation modes. In the first one, the LFSR loop is closed so it behaves as a typical LFSR. This mode will be used to check the embedded tests. In the second operation mode, the output of the noise source (NS-A or NS-B) is XOR-ed with the feedback bit in order to use the LFSR as a post-processing block.

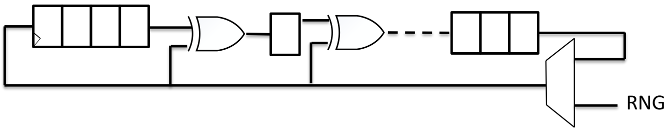


Fig. 7. Linear Feedback Shift Register

5) Control Operating Conditions: **(Inputs: On-Chip-Sensors; Outputs: External_Alarm).** As aforementioned, ROs are quite sensible to variations on operating conditions such as voltage and temperature[27]. We have decided to control these parameters in order to guarantee the validity of the stochastic model. To that end, as the TRNG has been implented in a FPGA, we have selected the XADC IP provided by Xilinx [20] due to its straightforward implementation. This block makes use of the on-chip sensors in order to measure different magnitudes (Temperature, V_{CCINT},V_{CCAUX} and V_{BRAM}.). If a measurement of an on-chip sensor lies outside the specified limits, then an alarm is set. We have established the limits according to the vendor recommendations for the FPGA (Artix-7). Other techniques as measuring ROs frequencies or those proposed in [9] can be considered as alternatives methods.

6) Checker + Controller : This block consists of 4 FSMs that control the different blocks. These FSMs have been implemented using one-hot encoding and the self-recovery state is set to the most restricted one in each case.

a) Tests FSMs: **(Inputs: Counters; Outputs: Fail_A, Fail_B, Error).** The purpose of these FSMs is to guarantee the correct operation of the different tests. To that end, we have implemented two FSMs: $FSM_{Entropy\_Tests}$ and FSM_{FIPS}. Each one receives and compares the outputs (counter values) of the duplicated tests. Once a fault is detected (counter values are not equal), an off-line testing mode starts. This off-line testing mode makes use of the LFSR to generate a known sequence of bits. The final response of both tests (entropy and FIPS tests) for this known sequence has been precomputed and stored in the FPGA. Finally, the precomputed value is compared to the test results and the faulty test is disconnected. At this point, as only one test is functional (A or B), the off-line testing mode will be used for each round of generated bits (8192 for the Entropy Tests and 20000 bits for the FIPS tests). If both tests fail, a final error state is reached. It is important to remark that while the TRNG is on the off-line testing mode, the TRNG is stopped. In Fig. 8 a simplified state diagram for both FSMs is shown.

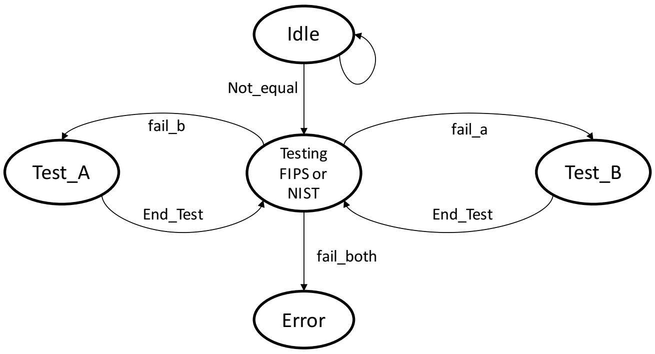


Fig. 8. Test FSMs diagram .

b) Noise source FSM : **(Inputs: Low_entropy_Alarm, Medium_entropy_alarm, External_Alarm; Outputs: Select_NS, Enhance_NS, Enhance_PP, Error).** This FSM receives the minimum entropy range computed in the entropy tests and handles the noise source. The alarm signal generated by the on-chip sensors is also taken into account. Fig. 9 shows a simplified state diagram for this FSM where two scenarios are considered.

In the first scenario, a faulty behaviour is discovered in the noise source thanks to the entropy tests, which can mean either a slight degradation of the minimum entropy (medium entropy alarm set) or an unacceptable degradation (low entropy alarm set). In the first case, the noise source will be reset (transient ROs shut-down) and an enhancement in the post processing will be carried out (signal *enhance_post-processing*). If the degradation persists, 128 extra ROs of the same noise source will be activated in order to contribute to the randomness. In addition, an improvement of the post-processing will be demanded. If this situation lasts, the noise source will be switched and the maximum post-processing correction will be required. Finally, an error state that requires the user attention is reached. It is important to note that the TRNG operation is not interrupted in this case. In the case of a high degradation on the entropy (low entropy alarm set), the procedure will be the same but the TRNG will be stopped until it reaches an acceptable minimum entropy.

In the second considered scenario, i.e., no fault has been detected but one of the on-chip sensors has sampled a value which falls outside the limits, the active noise source will enhance its response using the available 256 ROs and the post-processing will be increased in 2 orders. With these counteractions we will try to anticipate a possible fault.

If after a fault or a change in the surrounding conditions, the normal behaviour is maintained during 10 entropy test executions, the previous state will be reached.

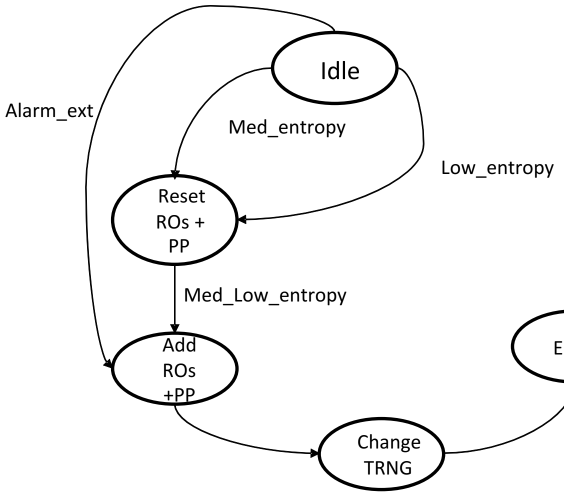


Fig. 9. Noise Source FSM diagram.

c) Post-processing FSM: **(Inputs: FIPS_Alarm, Enhance_PP ; Outputs: Select_PP, Select_Order, Error).** This FSM will handle the different post processing blocks (PP-A, PP-B and the LFSR). If the FIPS alarm is set to 1 or the signal *enhance post-processing* (generated by the Noise Source FSM) is set, the FSM will evolve to the next state. If the 130^{th} order filter cannot correct the fault, PP-B will be used to increase the entropy per bit of the final output. If the parity filter is not enough to obtain an acceptable output, the LFSR will be used as a post-processing block (mode 2). Ultimately, an error state will be reached. As in the Noise Source FSM, a counter is set in order to go back to the previous state if a normal behaviour is maintained during 10 FIPS tests executions. Fig. 10 presents the different states for this FSM.

It is important to note that all the FSMs reach a final state of Error. This state requires the user attention in order to recover the normal TRNG operation.

7) Anti-aging mechanism: **(Outputs: Select_NS).** ROs generate a high electronic activity during the random number generation. This switching activity during a long operation term will enhance aging effects, that can provoke frequency variations and may degrade the TRNG output. In order to guarantee that all ROs will age in the same way, we have introduced a counter that switches the noise sources (from A to B and vice-versa) every 1000 generations (i.e., 20 Mb). In this way, frequency variations of the two redundant versions of the noise source will be maintained as equal as possible.

IV. Experimental results

A. Hardware Resources

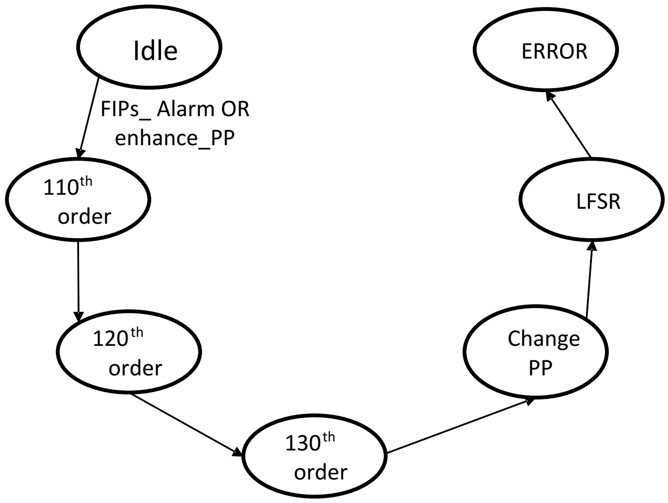


Fig. 10. Post-processing FSM diagram.

TABLE I
Logic Resources.

Element	Sunar[2]+ Wold[15]	Sunar + Wold + Tests	Our Proposal
Slice LUTs	**252**	**1212**	**2645**
Slice Registers	**150**	**228**	**1286**
DSPs	**0**	**1**	**2**
XADC	**0**	**0**	**1**
BRAM	**0**	**20 Kb**	**20 Kb**

In this subsection we analyse the impact of the TRNG enhancement in the main metrics: Logic resources, Power consumption and Throughput. The target FPGA is an Artix-7 35T.

1) Logic resources: Table I shows the LUTs, registers and DSPs for different TRNG implementations. The first column shows the modified implementation of Sunar et al. where only the noise source (128 ROs) and the post-processing are included. The second column shows the hardware resources used in an implementation that includes the entropy tests and the FIPS tests. Finally, the third column shows the results of our enhanced proposal, which includes redundant blocks.We have included 20 Kb of BRAM memory in order to store the generated random numbers until all the test are executed (randomness guarantee). These BRAM blocks are optional,as the data could be stored outside from the FPGA, and validated or discarded after receiving the corresponding error signals.

As expected, our enhanced proposal uses more resources than the other two implementations. For the purpose of a fair comparison, the first implementation (modified Sunar et al. TRNG) is not compared quantitatively because almost all the TRNG standards require to test somehow the generated bitstream [4][3]. Comparing the modified Sunar et al. implementation with testing capabilities with our proposal, it can be appreciated the impact of the redundant blocks in the resources. More specifically, the number of LUTs is roughly doubled and 5.6 times more registers are used. This increase is mainly due to the redundant blocks but also to the additional FSMs, the cut-off values that must be stored and the LFSR block. More specifically, the 40% of the resources are devoted

TABLE II
POWER CONSUMPTION.

Power	Sunar[2]+ Wold[15]	Sunar + Wold + Tests	Our Proposal
Dynamic	16 mW	60 mW	116 mW
Static	63 mW	75 mW	118 mW

TABLE III
THROUGHPUT.

Sunar[2]+ Wold[15]	Sunar + Wold + Tests	Our Proposal
3 Mbps	2.994 Mbps	2.994 Mbps

to the noise sources. The 9% of the resources are destined to the post-processing blocks (8%) and the LFSR (1%). Entropy and FIPS tests use 8% and 22% respectively. Finally, the checker and controller use a 19% of the total resources. It is important to remark that our proposal uses 12.71% of the available LUTs and a 3% of the flip-flops.

2) Power consumption: Table II shows the power consumption estimations provided by the Vivado tool. We have selected the default settings for the environment, power supply and switching activity. The values obtained are comparable to those published in [23].

As expected, our enhanced proposal consumes almost the double than the original proposal. It is noteworthy the high percentage that represents the dynamic power consumption.

3) Throughput: The throughput presented in Table III has been obtained for a sampling frequency of 300 MHz. A best case scenario has been considered where only a 100^{th} order filter is used as post-processing. The noise source of the three implementations generates one raw random bit per clock cycle. For the original implementation, this throughput will only be penalized by the post-processing. The other two implementations will generate blocks of 20000 random bits (FIPS dataset length) each 20 ms and the tests will introduce a latency of around 100 clock cycles. It is important to remark that all the executed tests are on-the-fly tests that are executed while the data is generated. The 100 clock cycles are necessary in order to carry out the final computations of the FIPS tests.

It is also interesting to compare the throughput of our proposal for three different scenarios where the output is still random: the best case scenario is the aforementioned scenario where the post-processing needs are the minimum. We have defined an intermediate scenario where the maximum filter order is required. The worst case scenario considers the failure of one entropy test block, one FIPS test block and both post-processing blocks. In this scenario, as defined in previous sections, the off-line testing mode will be activated for both tests after each execution. In addition the LFSR will be used in the post-processing. In Table IV the results for the different scenarios are presented.

TABLE IV
THROUGHPUT OF OUR PROPOSAL FOR DIFFERENT SCENARIOS.

	Best Case	Intermediate case	Worst case
Throughput	2.994 Mbps	2.307 Mbps	0.360 Mbps

TABLE V
FAULT INJECTION CAMPAIGNS SUMMARY.

Type	bit flip	stuck-at
Injected Faults	15000	53152
Masked	3682 (24.54%)	25768 (48.47%)
Detected	11318 (75.45%)	27384 (51.53%)
Random	199 (1.31%)	0
Not-Random	0	0

B. Fault Injection for Dependability Validation

Fault injection is an effective and widely used method for test, assessment and dependability benchmarking of fault-tolerant and fail-safe systems. We have used LIFTING[6] in order to evaluate the system dependability. LIFTING is an open source simulator able to perform both logic and fault simulation for stuck-at faults and single event upset (SEU) on digital circuits described in Verilog. In addition, this fault simulator provides many features for the analysis of the fault results.

In order to use this simulator, we have generated a Verilog description of the TRNG deterministic parts (Noise source and XADC IP are not evaluated). We have created a 200000 random bits test bench in order to stimulate the circuit. Two fault injection campaigns have been simulated: one for injecting bit-flips and one for single stuck-at faults.

TableV summarizes the results of the different fault injection campaigns. The first and second rows represent the type of fault and the number of injected faults respectively. The third row represents the faults that have been masked (no alarm and no change at the output). The fourth row presents the number of faults that have been detected (raise an alarm). The fifth row -random- shows the number of undetected faults that change the output but do not have any impact in the statistical quality of the output. Finally, row six presents the undetected faults that have an impact in the statistical distribution of the output (Not-Random).

- **Bit-flip fault injection campaign:** In order to obtain statistically significant results, we adopted the method proposed in [26].

$$n = \left(\frac{N}{1 + e^2 \cdot \frac{N-1}{t^2 \cdot p \cdot (1-p)}} \right) \qquad (2)$$

The number of samples is expressed in Equation 2, where:

  - n is the number of faults to inject;
  - N is the overall number of possible injected faults;
  - p is an estimation of the value being searched;
  - e is the expected margin of error;
  - t is the expected confidence level.

Since the overall number of possible faults is extremely high, we compute the limit of Equation 2 for N tending to infinite. We also consider the conservative result by setting p equal to 0.5. The result is shown in Equation 3.

$$n = \lim_{N \to \infty} \left(\frac{N}{1 + e^2 \cdot \frac{N-1}{0.25 \cdot t^2}} \right) = \frac{t^2}{4 \cdot e^2} \qquad (3)$$

We consider for all experiments a margin of error of 1% with a confidence level of 98%, leading to 15000 fault injections per program. The results of this injection show that the proposed enhanced TRNG is able to self-recover from different fault scenarios. The 24,54% of the injected faults have been masked. These masked faults appear because they have been injected in a block that is not used in that moment or they have been injected in the FSMs with self recovery, so no effects appear at the output. The vast majority of the injected faults have set off an alarm (75,45%). Only a 1,31% of undetected faults without an impact in the statistical quality of the output are observed. In this case, all the faults were injected in the post-processing block so only one bit of the output changes. However, these bit-flips do not jeopardize the TRNG security. Indeed, the post-processing block is composed of 100 bits. Even if one bit is altered, the overall randomness is not affected since the other 99 bits will be random. Finally, there is no fault undetected that compromises the TRNG quality. It is important to note that the 38% of the faults have been detected instantly while the other 62% have been detected by the on-the-fly tests and the checker block.

- **Stuck-at fault injection campaign:** In this campaign, an exhaustive simulation that covers the stuck-at 0/1 of all the nodes on the circuit has been carried out. The injection of 53152 faults have been simulated. In this case and for the same reasons than in the bit-flip injection campaign, the 48,47% of the faults have been masked. The rest of the faults (51,53%) have raised an alarm. As expected, in this simulation, all the faults that affect the output have been detected because their effects are more noticeable than a simple bit-flip.

It is important to remark that in both fault injection campaigns, all the faults affecting test blocks have been detected, so we can guarantee that if not alarm is raised, all the generated numbers will have a minimum entropy rate. In addition, it is noteworthy that no faulty output (Not-Random) has been detected in the aforementioned fault injection campaigns. In addition to these faults, our proposal also offers protection against fluctuations on the operation conditions, clock glitches and aging protection.

In order to show the importance of redundant logic, we have compared our results with the non-redundant TRNG implementation (modified Sunar et al. + Tests). We have obtained the number of undetected faults for this implementation. In the bit-flip campaign, the 44.38% of the faults injected will pass unnoticed. On the other hand, for the Stuck-at fault campaign the 59,87% of the faults will be undetected. Basically, these faults are concentrated in the test blocks that are not checked. These results demonstrate that, despite the increase in area and power consumption, redundancy is crucial to assure dependability.

V. Conclusions

In this work we have addressed some of the challenges that TRNGs pose regarding dependability. TRNGs are of paramount importance in many cryptographic systems, so their dependability must be assured. This means that all the TRNG blocks (noise source, post-processing and tests) must be dependable. The scientific community has focused its efforts into increasing the entropy level without paying attention to the system dependability. To the best of our knowledge, this is the first work where the entire system dependability is addressed.

To that end, we have proposed a number of general guidelines that can be applied to TRNG systems in order to increase their dependability. These techniques include well-known hardening methods which were adapted for TRNGs. In addition, we have presented some specific recommendations that are not typically considered in general-purpose digital circuits (temperature control, anti-aging mechanisms,etc).

A case of study has been presented in order to verify the validity of the aforementioned guidelines. More specifically, we have implemented an enhanced version of the TRNG presented by Sunar et al. [2] and later modified by Wold et al.[15]. The flexibility of this proposal has allowed us to apply different methods in order to increase the system dependability. The applied techniques are primarily aimed to guarantee a minimum entropy level and to offer self-recovery capabilities from different scenarios including attacks that cause bit-flips and stuck-at faults. Among the applied techniques stand out the redundancy applied to several blocks (tests, post-processing, noise source) and the *checker+controller* block that manages the different situations in order to optimize the response for different faulty states.

We have presented the overhead introduced by the enhancement. In terms of area, power consumption and throughput the results show that our proposal is affordable for many applications. Finally, we have carried out two fault injection campaigns (bit-flip and stuck-at faults) where the dependability of our proposal has been tested. No faults that affect the statistical distribution of the output have passed undetected. We can conclude that, with an affordable overhead, we have increased the system dependability being able to counteract many attack situations.

Acknowledgements

This work was partially supported by ICT COST Action IC1204 and by the Spanish Ministry of Economy and Competitiveness under contract ESP2015-68245-C4-1-P.

References

[1] Fips pub 140-2, security requirements for cryptographic modules, 2002. U.S.Department of Commerce/National Institute of Standards and Technology.

[2] B. Sunar, W. J. Martin and D. R. Stinson . A provably secure true random number generator with built-in tolerance to active attacks. *IEEE Transactions on Computers*, 56:109–119, 2007.

[3] E. Barker J. Kelsey. Recommendation for the entropy sources used for random bit generation. *NIST DRAFT Special Publication 800-90B*, 2012.

[4] W. Schindler and W. Killmann. Evaluation Criteria for True (Physical) Random Number Generators Used in Cryptographic Applications. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '02, pages 431–449, London, UK, UK, 2003. Springer-Verlag.

[5] A. Avizienis, J-C. Laprie, B. Randell and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, January 2004.

[6] A. Bosio and G. D. Natale. Lifting: A flexible open-source fault simulator. In *2008 17th Asian Test Symposium*, pages 35–40, Nov 2008.

[7] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray and S. Vo. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. *Technical Report*, 2001.

[8] B. Yang, V. Rocic, N. Mentens and I. Verbauwhede. On-the-fly tests for non-ideal true random number generators. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2017–2020, May 2015.

[9] B. Yang, V. Rozic, W. Dehaene, N. Mentens and I. Verbauwhede. TOTAL: TRNG On-the-fly Testing for Attack detection using Lightweight hardware. In *Design, Automation and Test in Europe (DATE 2016)*, page 6, Dresden,GE, 2016. IEEE.

[10] M. Cassel and F.L. Kastensmidt. Evaluating one-hot encoding finite state machines for seu reliability in sram-based fpgas. In *IOLTS*, pages 139–144. IEEE Computer Society, 2006.

[11] D. Boneh , R. A. Demillo and R. J. Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14:101–119, 2001.

[12] E. Bohl, M. Lewis and S. Galkin. A true random number generator with on-line testability. In *2014 19th IEEE European Test Symposium (ETS)*, pages 1–6, May 2014.

[13] F. Lu, G. D. Natale, M-L. Flottes and B. Rouzeyre. Simulating Laser Effects on ICs, from Physical Level to Gate Level: a comprehensive approach. In *TRUDEVICE'2014: Test and Fault Tolerance for Secure Devices*, Paderborn, Germany, May 2014.

[14] H. Martin and T. Korak and E. S. Millan and M. Hutter. Fault attacks on strngs: Impact of glitches, temperature, and underpowering on randomness. *IEEE Transactions on Information Forensics and Security*, 10(2):266–277, Feb 2015.

[15] K. Wold and C. H. Tan. Analysis and enhancement of random number generator in fpga based on oscillator rings. In *2008 International Conference on Reconfigurable Computing and FPGAs*, pages 385–390, Dec 2008.

[16] J. C. Laprie. Dependable computing and fault tolerance : Concepts and terminology. In *Proc. 15th IEEE Intl Symp. Fault-Tolerant Computing (FTCS-15)*, pages 2–11, Jun 1985.

[17] M. Drutarovský and M. Varchola and O. Vancák. Optimized FIPS 140 statistical tests IP core embedded in FLASH based ACTEL FPGA. In *Proceedings of the 53-th Internationales Wissenschaftliches Kolloquium*, pages 1–5,, Ilmenau, Germany, September 7–10, 2009.

[18] M. Shafique, M. Khan, K. M. Usman, O. Tüfek and J. Henkel. Enaam: Energy-efficient anti-aging for on-chip video memories. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pages 101:1–101:6, New York, NY, USA, 2015. ACM.

[19] M. T. Rahman, K. Xiao, D. Forte, X. Zhang, J. Shi and M. Tehranipoor. Ti-trng: Technology independent true random number generator. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2014.

[20] C. Murphy. Driving the xilinx analog-to-digital converter (xapp795). In *Application Note: 7 Series FPGAs*, Feb 2015.

[21] N. Bochard, F. Bernard, V. Fischer, B. Valtchanov. True-randomness and pseudo-randomness in ring oscillator-based true random number generators. *International Journal of Reconfigurable Computing*, 2010, 2010.

[22] O. Kömmerling and M. G. Kuhn. Design principles for tamper-resistant smartcard processors. In *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, WOST'99, pages 9–20, Berkeley, CA, USA, 1999. USENIX Association.

[23] O. Petura, U. Mureddu, N. Bochard, V. Fischer and L. Bossuet. A survey of ais-20/31 compliant trng cores suitable for fpga devices. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–10, Aug 2016.

[24] P. Bayon, L. Bossuet, A. Aubert and V. Fischer. Fault model of electromagnetic attacks targeting ring oscillator-based true random number generators. *Journal of Cryptographic Engineering*, 6(1):61–74, 2016.

[25] P. Maistri, R. Leveugle, L. Bossuet, A. Aubert, V. Fischer, B. Robisson, N. Moro, P. Maurine, J. M. Dutertre and M. Lisart. Electromagnetic analysis and fault injection onto secure circuits. In *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6, Oct 2014.

[26] R. Leveugle, A. Calvez, P. Maistri and P. Vanhauwaert . Statistical fault injection: Quantified error and confidence. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE, 2009*, pages 502–506.

[27] R. Santoro, O. Sentieys and S. Roy. On-the-fly evaluation of fpga-based true random number generator. In *2009 IEEE Computer Society Annual Symposium on VLSI*, pages 55–60, May 2009.

[28] B. Razavi. *Fundamentals of Microelectronics*. Wiley, 2008.

[29] T. A. Markettos and S. W. Moore. The frequency injection attack on ring-oscillator-based true random number generators. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '09, pages 317–331, Berlin, Heidelberg, 2009. Springer-Verlag.

[30] T. Figliolia, P. Julian, G. Tognetti and A.G. Andreou. A true random number generator using rtn noise and a sigma delta converter. volume 2016-July, pages 17–20, 2016.

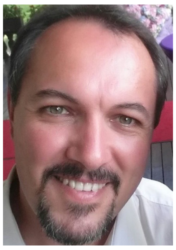
[31] V. Fischer and D. Lubicz. Embedded Evaluation of Randomness in Oscillator Based Elementary TRNG. In L. Batina and M. Robshaw, editors, *Cryptographic Hardware and Embedded Systems CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 527–543. Springer Berlin Heidelberg, 2014.

[32] V. Rocic, B. Yang, N. Mentens and I. Verbauwhede. Canary numbers: Design for light-weight online testability of true random number generators. Cryptology ePrint Archive, Report 2016/386, 2016.

[33] P.Z. Wieczorek. Lightweight trng based on multiphase timing of bistables. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(7):1043–1054, 2016.

[34] 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*, PP(99):1–12, 2016.

Honorio Martin received the Ph.D. degree in electronics engineering, from Universidad Carlos III de Madrid, Spain, in 2015. He is a Postdoctoral Researcher with the Department of Electronic Technology, Universidad Carlos III de Madrid, Madrid, Spain. His research interests include the study of lightweight cryptography, hardware implementations, radio-frequency identification systems, and low-power designs.



Giorgio Di Natale received the PhD in Computer Engineering from the Politecnico di Torino (Italy) in 2003 and the HDR (Habilitation Diriger les Recherches) in 2014 from the University of Montpellier II (France). He is currently Director or Research for the National Research Center of France at the LIRMM (Laboratoire dInformatique, de Robotique et de Microlectronique de Montpellier - UMR 5506), a joint research laboratory between the CNRS and the University of Montpellier. His research interests include hardware security and trust, secure circuits design and test, reliability evaluation and fault tolerance, software implemented hardware fault tolerance, and VLSI testing. He has been involved in projects funded by the EU, Italy and France. He has been the scientific leader of the FP7 CLERECO project for the CNRS partner (2013-2016). From 2012 to 2016 he was also the action chair of the COST Action TRUDEVICE (Trustworthy Manufacturing and Utilization of Secure Devices), the biggest European research network on hardware security and trust. He is the chair of the European section of the TTTC, Golden Core member of the Computer Society and Senior member of the IEEE.

Luis Entrena received the Industrial Eng. degree from Universidad de Valladolid (Spain) in 1998 and the PhD degree in Electronic Engineering from Universidad Politcnica de Madrid (Spain) in 1995. From 1990 to 1993 he was with AT&T Microelectronics at Bell Labs (USA). From 1993 to 1996 he was Technical Project Leader at TGI (Spain). Since 1996 he is Associate Professor at Universidad Carlos III de Madrid (Spain), where he has served as Head of the Electronic Technology Department and Director of the Postgraduate Program in Electrical Engineering. He has co-authored over 150 papers and one patent. His current research interests include on-line testing, fault tolerance, soft error sensitivity evaluation and mitigation, hardware security and hardware acceleration.