A Novel Pseudo Random Number Generator Based on L'Ecuyer's Scheme

Francesco Buccafurri and Gianluca Lax
DIIES Dept., University of Reggio Calabria, Reggio Calabria, Italy

Keywords: Pseudo Random Number Generator, L'Ecuyer's Scheme, Cryptographic Applications.

Abstract: In this paper, we propose a new lightweight L'Ecuyer-based pseudo random number generator (PRNG). We show that our scheme, despite the very simple functions on which it relies on, is strongly secure in the sense that our number sequences pass the state-of-the-art randomness tests and, importantly, an accurate and deep security analysis shows that it is resistant to a number of attacks.

1 INTRODUCTION

A pseudo random number generator (PRNG, for short) based on L'Ecuyer's scheme (L'Ecuyer, 1994) is able to deterministically generate, starting only from an initial secret seed, a sequence of numbers which is indistinguishable from a true random sequence and such that there is no way for the attacker to predict the future output by knowing past subsequences of outputs. The scheme is composed of (1) a constant transition function that maps any inner state s_i to the successive state s_{i+1} only on the basis of s_i (with no other input) and (2) an output function g such that the i-th element of the pseudo-random sequence is computed as $u_i = g(s_i)$. PRNGs have a lot of applications, such as the generation of the keystream of a stream cipher, the generation of keys of block ciphers, the implementation of protocols of strong authentication and so on.

Typically, PRNGs use cryptographic functions such as ciphers or one-way functions (Blum et al., 1986; Li and Zhang, 2005; Cox et al., 2011). Obviously, this introduces a certain degree of computational complexity. As a matter of fact, in many situations, it would be desirable to reduce as much as possible the overall computational effort of the device, anyway keeping high the security level of adopted algorithms. This is for example the case of cryptography-based applications in wireless sensor networks (Tang et al., 2004; Alcaraz and Lopez, 2010; Wang, 2011) or in wireless devices (mobile phones, WLANs, etc.), where the efficiency of algorithms is required both for real-time strict limitations and for minimizing power consumption. Observe that there exist a number of PRNGs, like those presented in (Melià-Seguí et al., 2013; Dolev et al., 2011; Huang et al., 2010), specifically designed for EPC Gen2 standard (EPCglobal, 2004); however, their use is limited to this scope (where, for example, random numbers are 16-bit wide).

In this paper, starting from an initial idea presented in the e-commerce setting (Buccafurri and Lax, 2011), by improving it and testing its security, we propose a lightweight PRNG based on L'Ecuyer's scheme which relies on very simple functions easily implementable in hardware for the transition function and a slight modification of CRC (Hill, 1979) for the output function. Thus, our PRNG does not make use of cryptographic functions in favor of computational efficiency.

We show that number sequences generated by our scheme pass the most known state-of-the-art randomness tests (National Institute of Standards and Technology, 2014); moreover, a preliminary security analysis does not disclose any weakness w.r.t all the attacks we can hypothesize. In other words, we argue that our schema is compliant with the security requirements of the German Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik, 2014).

This position paper represents a first checkpoint of our research aimed at (1) presenting the new PRNG, (2) highlighting that it is is very cheap from a computational point of view for the reasons illustrated above, and (3) giving a first solid argumentation about its security (proof of full randomness and partial analysis on resistance to cryptanalysis attacks). The next step is to deepen the study of the PRNG security against

Buccafurri F. and Lax G..
A Novel Pseudo Random Number Generator Based on L'Ecuyer's Scheme.
DOI: 10.5220/0005050303210328
In *Proceedings of the 11th International Conference on Security and Cryptography* (SECRYPT-2014), pages 321-328
ISBN: 978-989-758-045-1

cryptanalysis attacks and to compare our PRNG with existing PRNGs on the aspect of efficiency.

The structure of the paper is the following. In the next section, we define the notation used in the paper. In Section 3, we provide the detailed definition of the elements composing our scheme. We analyze the security of our scheme both theoretically and experimentally in Section 4. In Section 5, we deal with the computational costs of our PRNG. Finally, in Section 6, we draw our conclusions.

2 NOTATION

In order to define our scheme, we need the following notations.

- We denote by $x^k = (x_1, \ldots, x_k)$ a k-bit string, where x_j, with $1 \leq j \leq k$, represents the j-th bit numbered from left to right.
- We denote by $\overleftarrow{x^k} = (x_k, \ldots, x_1)$ the reverse string of x^k.
- Given a positive integer p, $x^k + p$ denotes the k-bit string representing the number obtained by summing (in 2^k-modulo arithmetic) x^k thought as a binary number and p. For example, given $p = 1$ and $x^3 = 111$, $x^3 + 1 = 000$, since $(111 + 001)(\textbf{mod}\ 1000) = 000$.
- Given a k-bit string x^k, we denote by $[x^k]_{i,j}$ with $1 \leq i \leq j \leq k$ the sub-string of x^k obtained by keeping the $j - i + 1$ bits starting from the i-th left-most bit. For example, given $x^k = 1000$, $[x^k]_{1,2} = 10$.
- Given a k-bit string x^k, we denote by $[x^k]_i$ with $1 \leq i \leq k$ the i-th left-most bit, i.e., $[x^k]_{i,i}$.
- We denote by $x^i x^j$ the $(i + j)$-bit string obtained by appending x^j to x^i.
- Given a k-bit string x, we denote by $\overrightarrow{x}$ the k-bit string obtained from x by circularly right shifting it as many times as the number of 1s occurring in it. For example, given $x = 1100$, then $\overrightarrow{x} = 0011$.
- Finally, we denote by 1^k (0^k, resp.) the k-bit string composed of all 1s (0s, resp.).

3 PRNG SCHEME

In this section, we define our PRNG that adopt a 1023-bit states and generates 128-bit numbers. Our PRNG is based on the L'Ecuyer definition (L'Ecuyer, 1994). Thus, it consists of a tuple $\langle S, T, O, g, s_0 \rangle$ where S is the finite state set, $T : S \to S$ the transition function, O the output space, $g : S \to O$ the output function, and $s_0 \in S$ the (initial) seed of the generator. Starting from the initial state s_0 and using the transition function, the PRNG produces a chain of states $s_0, s_1, s_2, \ldots$ such that $s_i = T(s_{i-1})$ for each $i \geq 1$. From each state of this chain, say s_c, it is possible to compute $x_c = O(s_c)$ which is the c-th random number generated starting from the initial seed s_0.

States of our PRNG consists in 1023-bit strings, so that $|S| = 2^{1023}$. The transition function T is parametric with respect to a positive odd integer m and is defined as follows.

Definition 3.1. *Given a k-bit string s^k, with $k = 1023$, we define $T(s^k) = \overleftarrow{s^k} + m$.*

In words, $T(s^k)$ is obtained by reversing the string s^k and, then, by summing m (modulo 2^k).

Obviously, the first requirement for a good L'Ecuyer PRNG is that the period of the function T is as large as possible, hopefully 2^k (the upper bound). Theorem 4.1 in Section 4.4 proves that the function T of our PRNG has maximum period (i.e., 2^k).

The output function is used to produce a random number x from the current state s, with the requirement that the knowledge of x does not give an attacker the possibility of guessing s. To do this, a one-way function, such as a cryptographic hash function, can be adopted (as it often happens in state-of-the-art PRNG). With the purpose of saving computational effort, we observe that we can be satisfied also by a non-cryptographic hash function whose inversion is computational feasible, but the number of average colliding states generating the same output is so big that it is infeasible to guess the actual state (corresponding to the observed output). This principle can be profitable applied in our case if the knowledge of a colliding state does not give the attacker any advantage, and the weakness of the hash function does not allow the attacker to apply cryptanalysis-based attacks.

Following the above approach, we implement the output function as a modified version of CRC (Cyclic Redundancy Check) (Hill, 1979), which is a non-cryptographic hash function widely used in error-detection context.

Classical CRC is computed to produce a n-bit string, named *checksum*, starting from an arbitrary length string, called *frame*, such that also a slight change of the frame produces a different checksum. The checksum is computed as the rest of the binary division with no carry bit (it is identical to XOR), of the frame, by a predefined *generator polynomial*, a $(n+1)$-bit string representing the coefficients of a polynomial with degree n. CRC is thus parametric w.r.t. the generator polynomial and for this reason

there are many kinds of CRCs. For example, the most frequently used are CRC32 or CRC64, which generate a checksum of length 32 and 64 bits, respectively. Obviously, the higher the checksum length, the better the effectiveness of CRC in error-detecting is. In our PRNG, we use a 128-bit CRC.

Observe that, given a k-bit frame s^k and its w-bit (with $k > w$) checksum c^w computed by CRC, there exist 2^{k-w} (colliding) k-bit strings s_i^k such that $\text{CRC}(s_i^k) = c^w$. Moreover, its implementation easiness and efficiency make CRC very appealing to be used in this context.

Besides these nice features, CRC is not immune from malicious attacks exploiting its linearity w.r.t. XOR. As a consequence, we have modified CRC design keeping the nice computational features of CRC but eliminating its weakness. The new CRC is obtained by applying r cyclic right shifts to the state before the standard CRC computation, where r is equal to the number of 1s occurring in the state itself.

We have seen that CRC is parametric w.r.t. the length of the generator-polynomial and the values of its coefficients. Consequently, the efficacy of CRC strictly depends also on the latter parameter.

Now we are ready to define our output function g (recall that we denote by $\overrightarrow{s_c^k}$ the k-bit string obtained from s_c^k by circularly right shifting it as many times as the number of 1s occurring in it).

Definition 3.2. *Given a k-bit string s_c^k, with $k = 1023$, then $g(s_c^k) = CRC128(\overrightarrow{s_c^k})$.*

Concerning the implementation of CRC adopted in our output function, we use the 128-bit CRC having coefficients set according to the ECMA standard (ECMA, 1992) (thus choosing a concrete application setting among others).

4 SECURITY ANALYSIS

In this section, we analyze the robustness of the proposed number generation scheme both by statistical analysis of randomness and by considering a number of possible strategies followed by an attacker to guess future output.

4.1 Randomness of the Generation Scheme

To check whether the output of our PRNG seems to be random, we used the National Institute of Standards and Technology statistical test suite, named NIST 800-22 (Rukhin et al., 2001), which consists of 16 tests focusing on a variety of different types of non-randomness that could exist in a sequence.

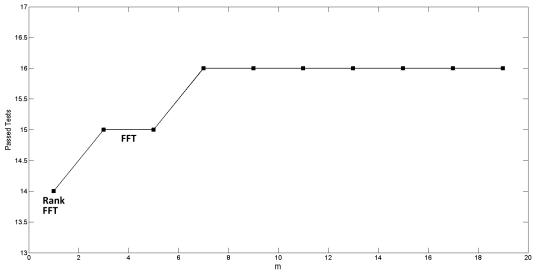

Figure 1: Passed tests for different values of m.

The results of our experimental campaign allow us to state that, according to the German Federal Office for Information Security (BSI) (Bundesamt für Sicherheit in der Informationstechnik, 2014), our PRNG is at least in class K2. Schemes in K2 generate random numbers having similar statistical properties to random numbers which have been generated by an ideal random number generator (Schindler, 1999).

In order to test our PRNG, we have generated 100 sequences of 10^6 bits as required by the suite (Rukhin et al., 2001) and then we have performed all the statistical tests. Each test is based on a calculated test statistic value, which is a function of the data. The test statistic is used to calculate a P-value representing the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested. If a P-value for a test is equal to 1, then the sequence appears to have perfect randomness. A P-value equal to zero indicates that the sequence appears to be completely non random. Typically, a significance level α is fixed for the test. As suggested by the suite, in our experiments, we fixed $\alpha = 0.01$, thus expecting one sequence in 100 sequences to be rejected by the test if the sequence was random.

In the first experiment, we test the randomness of the numbers produced by our PRNG varying the parameter m of the transition function, which, we recall, represents the number of increments of the state to be performed. In Figure 1, we report the overall number of tests passed by our PRNG for different values of m. Moreover, whenever this number is less than 16, we report also which tests failed.

From the analysis of this figure, we observe that even low values of m produce numbers with good randomness properties, only for $m \geq 7$, our PRNG is able to pass all the tests. In particular for $m = 1$, both Rank and FFT tests fail, while for $m = 3$ and $m = 5$ only FFT test is not passed. Recall that the purpose of the rank test is to check for linear dependence among fixed length substrings of the original sequence, while

FFT test is able to detect periodic features (i.e., repetitive patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness.

Now, we focus on the analysis of our PRNG when $m = 7$. We report in Table 1 the results obtained with the NIST test battery (Rukhin et al., 2001). When a test produces more than one P-value, we averaged them and marked the test by the symbol $^\diamond$. The minimum pass rate for each statistical test is about 0.96. From the analysis of these results, we can conclude that the output produced by our PRNG set $m = 7$ seems to be random.

We have performed another experiment aimed at showing that the randomness of our PRNG derives by the combination of the transition function T and the output function g. In particular, we have substituted in the scheme the function T with a simple counter function. Indeed, it is well known that full randomness can be obtained by applying a cryptographic hash function like SHA-1 to just a counter. But recall that our output function is a non-cryptographic hash function, so we cannot expect the same result. Indeed, the obtained sequences are very far from being random, as shown by the randomness tests whose results are reported in Table 2. In this table, we mark by the symbol * the six failed tests.

In the next sections, we provide a first analysis of the security of our PRNG under the cryptanalysis point of view. Such an analysis is necessary to argue that our schema is in class K3 and K4 according to the German Federal Office for Information Security.

4.2 CRC-Linearity-based Attack

It is well known that CRC is not immune from malicious attacks exploiting its linearity w.r.t. XOR. In particular, it holds that CRC(a XOR b) =CRC(a) XOR CRC(b), that is, the checksum of the XOR of two numbers is equal to the XOR of the checksums of the two numbers. In the case of our output function, which is based on CRC, this property could be in principle exploited by an attacker to obtain the output of the i-th state of an user (i.e., x_i =CRC(s_i^k)) starting from the knowledge of (1) the output of the j-th state of the user and (2) the XOR between s_i^k and s_j^k. Moreover, observe that the transition function operates a reverse of the string at each step just to introduce a suitable "noise", moving away its behavior from the pure XOR (that would allow the attack described above). The simple increment[1] (the simplest transition function that one could imagine) behaves exactly as a XOR every time the sum does not produce carry (i.e., every two steps). Unfortunately, it is easy to verify that the introduction of the reverse operation, even though beneficial, is not enough. Indeed, every two steps, the "noise" introduced by the reverse operation is *quasi*-cancelled. We use the prefix *quasi* because the transition function includes also the increment at each step.

To understand how this could be exploited for an attack, we observe that when a state s_i^k has both the left-most and the right-most bit 0 (i.e., every four steps), the attacker knows that s_i^k XOR $s_{i+2}^k = 10^{k-2}1$ (recall that, according to our preliminary notations, $10^{k-2}1$ denotes a k-bit string of the form $1\cdots1$, with $k-2$ 0s). Thus, the CRC of s_{i+2}^k is easily predictable by exploiting the above property. This behavior can be generalized also for other bit configurations. It is easy to see that if s_i^k is of the form $00\cdots01$, then we expect that the XOR with the state generated two steps ahead is of the form $10^{k-3}11$. Again, if s_i^k is of the form $10\cdots00$, then we expect that the XOR with s_{i+2}^k is of the form $110^{k-3}1$. Finally, if s_i^k is of the form $10\cdots01$, then we expect that the XOR with s_{i+2}^k is of the form $110^{k-4}11$. This is a symptom of the alternating destructive effect of the reverse operation and, further, of the general invariance of the internal part of the state, when the transition function is applied. Observe that this negative effect is maximum whenever the state is palindromic, because the effect of the reverse is null also on a single step.

The next theorem gives us the probabilistic support that a quasi-random generation of the initial seed prevents this drawback for the entire life time of the PRNG in a real-life application.

Theorem 4.1. *Let t and k be two positive integers such that $t < 2^{\frac{k-4}{2}}$. Let s^k be a k-bit state of the form $10c^jd^{k-4-2j}e^j00$, where c^j and e^j are j-bit strings, d^{k-4-2j} is a $(k-4-2j)$-bit string containing at least one 0 and $j = \lceil log_2 t \rceil + 1$. It holds that the sequence $S^t = \langle s_0^k, \ldots, s_t^k \rangle$ such that $s_0^k = T(s^k)$ and $s_r^k = T(s_{r-1}^k)$ for $1 \leq r \leq t$ does not contain any state of the form $10f^{k-4}01$, where f^{k-4} is a $(k-4)$-bit string.*

Proof. The proof is omitted for space reasons. □

The theorem states that (i) fixing both the first and the last two bits of the initial seed (to 10 and 00, respectively), and (ii) ensuring that the seed contains an internal centered range whose bounds are distant $\lceil log_2 t \rceil + 1$ from the bottom (and the top) of the seed itself such that at least one 0 occurs in this interval, then it results that for at least t/m applications of the transition function (considering now any value for m),

[1] In favor of security, we set the step size of the transition function $m = 1$, which is obviously the most advantageous case for the adversary.

Table 1: Results obtained with the NIST battery.

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
9	6	6	19	7	11	7	8	15	12	0.055361	0.99	Frequency
13	11	6	9	8	15	14	13	5	6	0.202268	0.98	BlockFrequency
11	7	12	2	10	13	15	7	8	15	0.090936	1.00	CumulativeSums
12	11	8	11	7	10	10	11	9	11	0.987896	0.99	CumulativeSums
4	11	12	7	14	8	10	11	11	12	0.574903	0.99	Runs
9	7	11	9	13	9	9	12	11	10	0.971699	0.99	LongestRun
8	11	11	8	10	12	9	11	10	10	0.996335	0.98	Rank
15	11	7	15	8	7	9	6	8	14	0.275709	1.00	FFT
9	9	7	13	10	6	15	12	8	11	0.637119	0.99	NonOverlappingTemplate$^\diamond$
15	7	12	11	8	12	9	7	10	9	0.759756	0.97	OverlappingTemplate
9	12	9	12	10	11	8	13	10	6	0.911413	1.00	Universal
5	16	11	12	13	7	9	9	7	11	0.383827	0.98	ApproximateEntropy
5	6	5	7	4	5	8	9	10	5	0.706149	0.98	RandomExcursions$^\diamond$
4	4	3	6	7	12	4	10	5	9	0.122325	0.98	RandomExcursionsVariant$^\diamond$
9	9	7	10	13	10	9	13	7	13	0.851383	0.97	Serial
9	11	7	7	9	20	8	10	10	9	0.181557	0.98	Serial
11	5	7	15	11	9	8	12	11	11	0.616305	1.00	LinearComplexity

Table 2: Results obtained when T is replaced by the counter function.

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
6	8	4	12	7	10	12	11	13	17	0.153763	1.00	Frequency
6	14	19	10	6	7	6	7	11	14	0.035174	0.99	BlockFrequency
3	11	7	6	8	9	11	14	11	20	0.019188	1.00	CumulativeSums
2	10	5	6	12	7	11	6	15	26	0.000004	1.00	CumulativeSums
39	8	7	9	5	9	7	5	6	5	0	0.83	* Runs
11	13	8	9	10	7	11	13	11	7	0.883171	1.00	LongestRun
100	0	0	0	0	0	0	0	0	0	0	0.00	* Rank
100	0	0	0	0	0	0	0	0	0	0	0.00	* FFT
16	10	9	10	8	8	12	9	11	7	0.739918	0.97	NonOverlappingTemplate$^\diamond$
13	16	8	12	14	5	10	12	5	5	0.096578	1.00	OverlappingTemplate
13	10	10	14	5	6	5	12	16	9	0.153763	0.98	Universal
34	4	3	2	6	6	4	3	5	33	0	0.79	* ApproximateEntropy
10	4	5	7	8	8	4	6	5	3	0.602458	0.95	* RandomExcursions$^\diamond$
8	8	5	12	3	4	1	7	3	9	0.048716	0.98	RandomExcursionsVariant$^\diamond$
50	6	3	4	5	2	3	6	7	14	0	0.67	* Serial
34	12	3	4	3	4	5	3	11	21	0	0.82	* Serial
6	12	10	7	9	18	10	15	7	6	0.108791	1.00	LinearComplexity

we do not generate bad seeds (i.e., seeds of the form $10\cdots01$). For example, given $m = 7$, in order to have the above property for the first 2^{64} output numbers, we have to consider $t = 7 \cdot 2^{64} \approx 2^{67}$, and we have to set the initial seed to $10s_1^{65}s^{k-134}s_2^{65}00$, where s_1^{65}, s_2^{65} and s^{k-134} are randomly generated, with the only constraint that s^{k-134} contains at least one 0. It is easy to verify that the probability that a randomly generated string s^{k-134} does not satisfies the above requirement is $\frac{1}{2^{k-134}}$. Thus, the blind random generation could be also accepted, because, in our case in which $k = 1023$, this probability is $\frac{1}{2^{889}}$.

4.3 Palindrome-based Attack

As described in Section 3, our generation scheme needs an initial seed s^k. The natural way to set the initial seed is clearly its random generation. Nothing seems to dissuade from this simple and effective approach.

However, note that the reverse done by the *transition function* is vanished whenever the string is palindromic. This forces us to understand if a random generation of a seed can (probabilistically) result in such a bad situation. The next theorem shows that the probability of this event is actually negligible for sufficiently large k.

Theorem 4.2. *The probability that a randomly generated k-bit string, with k* **mod** $2 \neq 0$*, is palindromic is* $2^{-\frac{k-1}{2}}$.

Proof. First we prove the following claim. Claim 1.

The number of palindromic k-bit strings such that k is odd is $2^{\frac{k+1}{2}}$. We prove Claim 1 by induction on k (odd). *Basis ($k = 1$).* Trivial. *Induction ($k > 1$ and k **mod** $2 \neq 0$).* We assume that the claim holds for a $k > 1$ such that k **mod** $2 \neq 0$. We have to prove that it holds for $k+2$ too. The palindromic strings of length $k+2$ are of the form: either $0p^k0$ or $1p^k1$, where p^k denotes a palindromic k-bit string. As a consequence the number of palindromic strings of length $k+2$ is twice the number of palindromic strings of length k. The statement is thus proved. The theorem statement follows immediately from Claim 1, since the probability of occurrence of a palindromic string is $\frac{2^{\frac{k+1}{2}}}{2^k}$. □

On the basis of the above theorem, we can easily realize that for treatable values of k, the resulting probability is negligible. For example, if $k = 1023$, then the probability of having a palindromic initial seed is 2^{-511}.

4.4 Output-Observation-Based Brute Force Attacks

In this section, we examine the attack carried out by observing the output generated by the PRNG. As we have shown that cryptographic attacks cannot be carried out, in this section we consider brute force attacks.

The first attack we examine is the naive one: The adversary observes the output looking for some periodic repetition of generated numbers. To show that this attack is unfeasible it suffices to prove that the *periodicity* of the function $T(s^k) = \widetilde{s}^k + m$ is as large as possible, hopefully 2^k (i.e., the upper bound). This means that, starting from a k-bit string s^k, it is possible to generate $2^k - 1$ different new states before reobtaining s^k. This is guaranteed by the next theorem.

Theorem 4.3. *Given a k-bit string s_0^k with k **mod** $2 \neq 0$, let S^k be the sequence $\langle s_0^k, \ldots, s_{2^k-1}^k \rangle$ such that $s_i^k = T(s_{i-1}^k)$ for $1 \leq i \leq 2^k - 1$.*

Then it holds that $s_i^k \neq s_j^k$, for any i, j such that $0 \leq i < j \leq 2^k - 1$.

Proof. We proceed by induction on the length k of the strings.
Basis ($k = 1$). Trivial: the sequence R^1 is $\langle 0, 1 \rangle$.
*Induction ($k > 1$ and k **mod** $2 \neq 0$).* We assume that the theorem holds for a $k > 1$ such that k **mod** $2 \neq 0$. We have to prove that it holds for $k+2$ too. We start the proof by considering r_0^{k+2} as the $(k+2)$-bit string obtained by $0r_0^k0$. We proceed by computing the next values of r_0^{k+2}. By reversing r_0^{k+2} and by adding 1, we have $r_1^{k+2} = T(r_0^{k+2}) = (0\widetilde{r}_0^k0) + 1 = 0\widetilde{r}_0^k1$. Similarly, we obtain $r_2^{k+2} = (0\widetilde{r}_0^k1) + 1 = (1r_0^k0) + 1 = 1r_0^k1$. Now, after the inversion of r_2^{k+2} we obtain the string $1\widetilde{r}_0^k1$, and after the increasing, the last bit of the string becomes 0, with a carry bit to $\widetilde{r}_0^k$. Thus, $r_3^{k+2} = 1(\widetilde{r}_0^k+1)0 = 1r_1^k0$, since $r_1^k = T(r_0^k) = (\widetilde{r}_0^k+1)$. Again, we obtain that $r_4^{k+2} = (0\widetilde{r}_1^k1)+1 = 0(\widetilde{r}_1^k+1)0 = 0r_2^k0$, since $r_2^k = T(r_1^k) = \widetilde{r}_1^k + 1$. We can easily generalize the above reasoning to the first 2^{k+1} elements of R^{k+2}. In particular, for each r_i^{k+2} with $0 \leq i < 2^{k+1}$, we have that:

1. if i **mod** $4 = 0$, then $r_i^{k+2} = 0r_j^k0$
2. if i **mod** $4 = 1$, then $r_i^{k+2} = 0\widetilde{r}_j^k1$
3. if i **mod** $4 = 2$, then $r_i^{k+2} = 1r_j^k1$
4. if i **mod** $4 = 3$, then $r_i^{k+2} = 1r_{j+1}^k0$

where $j = (i/4) * 2$ and / denotes the integer division.

Now we have to characterize the remaining values of R^{k+2}. The last value generated by the rules above is $r_{2^{k+1}-1}^{k+2} = 1r_{(2^{k+1}-1)/4*2+1}^k0 = 1r_{2^k-1}^k0$. By inductive hypothesis, $r_{2^k-1}^k = 1^k$ (i.e., it is the k-string composed by all 1s). Thus, $r_{2^{k+1}-1}^{k+2} = 1^{k+1}0$, that is, it is composed by all 1s but the last right-most bit. After the inversion we obtain $\widetilde{r}_{2^{k+1}-1}^{k+2} = 01^{k+1}$ and after adding 1, it results that $r_{2^{k+1}}^{k+2} = 10^{k+1} = 1r_0^k0$, since $r_0^k = 0^k$. Then we compute $r_{2^{k+1}+1}^{k+2} = 1\widetilde{r}_0^k0 + 1 = (0\widetilde{r}_0^k1) + 1 = 0(\widetilde{r}_0^k+1)0 = 0r_1^k0$. Again, $r_{2^{k+1}+2}^{k+2} = (0\widetilde{r}_1^k0)+1 = 0\widetilde{r}_1^k1$ and $r_{2^{k+1}+3}^{k+2} = 1r_1^k1$. Thus, the general rules characterizing the last 2^{k+1} values of R^{k+2} are the following. For each r_i^{k+2} such that $2^{k+1} \leq i < 2^{k+2}$, we have that:

1. if i **mod** $4 = 0$, then $r_i^{k+2} = 1r_j^k0$
2. if i **mod** $4 = 1$, then $r_i^{k+2} = 0r_{j+1}^k0$
3. if i **mod** $4 = 2$, then $r_i^{k+2} = 0\widetilde{r}_{j+1}^k1$
4. if i **mod** $4 = 3$, then $r_i^{k+2} = 1r_{j+1}^k1$

where $j = (i - 2^{k+1})/4 * 2$. Observe that the last value is thus $r_{2^{k+2}-1}^{k+2} = 1r_{2^k-1}^k1$. By inductive hypothesis, $r_{2^k-1}^k = 1^k$. As a consequence, we have proved item (2) of the theorem. Let us prove now item (1). By construction, all 2^{k+2} elements of R^k differ each other at least either on the left-most bit, or on the right-most bit or on the sub-string r_i^k. This proves the item (1) of the claim.
Theorem 4.1 follows immediately from Claim 1. Indeed, we know that $r_0^k \in S^k$ since S^k contains all 2^k k-bit strings. Let i be such that $s_i^k = r_0^k$. To prove the statement we will show that each element of R^k is equal to an element of S^k and in particular that $r_a^k = s_b^k$ where $b = (a+i)$ **mod** 2^k. We will demonstrate this equality for (1) $0 \leq a < 2^k - i$, (2) $a = 2^k - i$ and

(3) $2^k - i < a < 2^k$. (1) By construction, $r_a^k = s_{a+i}^k$ for $0 \leq a < 2^k - i$. As a consequence, $r_{2^k-i-1}^k = s_{2^k-1}^k = 1^k$. (2) Since $T(r_{2^k-i-1}^k) = T(1^k) = 0^k$ we have that $r_{2^k-i}^k = s_0^k$. (3) By construction, $r_a^k = s_{a+i}^k$ for $0 \leq j < 2^k - i$. □

Now we prove that also the definition of g preserves the property of the transition function of generating 2^k different states. Thus, we have to guarantee that two different states do not collide into the same value after the shifting. The next theorem ensures this.

Theorem 4.4. *Given a sequence (of k-bit strings)* $S^k = \langle s_0^k, \ldots, s_{2^k-1}^k \rangle$ *such that* $s_i^k = T(s_{i-1}^k)$ *for* $1 \leq i \leq 2^k - 1$, *let* $\overrightarrow{S^k} = \langle \overrightarrow{s_0^k}, \ldots, \overrightarrow{s_{2^k-1}^k} \rangle$.

Then it holds that $\overrightarrow{s_i^k} \neq \overrightarrow{s_j^k}$, *for each i and j such that* $0 \leq i < j \leq 2^k - 1$.

Proof. We proceed by contradiction. Suppose that $\overrightarrow{s_i^k} = \overrightarrow{s_j^k}$ with $i \neq j$. Let u be the number of 1s in $\overrightarrow{s_i^k}$ (and, consequently, also in $\overrightarrow{s_j^k}$). Now, shifting both numbers by u left circular shifts, we obtain s_i^k and s_j^k, respectively, with $s_i^k = s_j^k$ by construction. Since this contradicts Theorem 4.1, it results that $\overrightarrow{s_i^k} \neq \overrightarrow{s_j^k}$. □

Now consider more sophisticated attacks based on the observation of the output. The most simple case is when the attacker knows just one output number, say PRN_i. At this point the attacker has to find the original state s_i such that $g(s_i) = \text{CRC128}(\overrightarrow{s_i}) = PRN_i$. Since the number of colliding states w.r.t. *CRC*128 is $2^{1023}/2^{128} = 2^{895}$, guessing one of these states is infeasible. Let check now what happens if the adversary randomly selects one of the above states. If the value $\overline{s_i}$ chosen by the attacker (among the 2^{895} states) differs from the actual s_i (i.e., the current state), then the probability that $g(T(\overline{s_i})) = g(T(s_i))$ is $\frac{1}{2^{128}}$. Observe that the above probability coincides with the probability of guessing a valid output number with no background knowledge.

Now consider the case the attacker knows a sequence C of c consecutive output numbers. By a brute force attack, the attacker should test $(\frac{2^{128}}{2})^c$ states to find a state $\overline{s}$ such that it produces such a sequence C. Observe that, since our generation scheme produces a mapping between a set of 2^{1023} strings and a set of 2^{128} numbers, by relying on oracles able to guess the used state among the 2^{128} ones, the adversary would be able to guess the future output if the length of the observed string is at least 8, since $1023/128 - 1 \approx 8$. However, the probability of having such oracles is $\sum_{i=1}^{8} 2^{-128} = 2^{-1024}$, which represents obviously an impossible event.

Algorithm 1: The Transition Function.

Input: $m \geq 1$ odd
Input: s^k
1: **for** $a = 1$ to m **do**
2: $b = 0$
3: **while** $[s^k]_{k-b} = 1$ **do**
4: $[s^k]_{k-b} = 0$, $b = b + 1$
5: **end while**
6: $[s^k]_{k-b} = 1$
7: **end for**
8: **for** $a = 1$ to $\lfloor k/2 \rfloor$ **do**
9: swap $[s^k]_a$ and $[s^k]_{k-a+1}$
10: **end for**
11: **return** s

5 COMPLEXITY ISSUES

In this section, we show that our PRNG is efficient in both the transition and the output function.

Concerning the transition function, we show by Algorithm 1 how to compute $s^k + p$ starting from s^k. An improved version of this algorithm will be presented after. The first one (Lines 1-7) produces p increments by 1 of the state. Each increment is achieved as follows. If the right-most bit is 0, then it is set to 1 and the single increment ends. Otherwise, it is set to 0 and the procedure is iterated to the left-most adjacent bit. The second block (Lines 8-10) performs the state reversing.

Concerning the second block, we observe that it can be omitted provided that a slight modification in the first block is implemented. Indeed, instead of reversing the state at each step, we can apply the increments alternatively on the left side (for the first, third, fifth, and so on, generated PRNs) and on the right side (for the second, fourth, and so on generated PRNs).

The next theorem shows that the implementation of the transition function is very efficient.

Theorem 5.1. *The amortized cost of the transition function algorithm is constant in the number of bits of the state.*

Proof. The algorithm requires to set to 1 either the right-most bit, when the last bit of the state is 0 (the probability that this case occurs is $1/2$), or the second last bit when the state ends by 01 (this has probability $1/4$), or the third last one if the string ends by 011 (the probability is $1/8$), and so on. The same occurs when increments are done on the left side. As a consequence, the amortized complexity of m increments is $\sum_{a=1}^{m} a \cdot \frac{1}{2^a} \leq 2$. □

Concerning the output function, we observe that CRC is widely used thanks to its efficiency. Indeed, CRC efficiency is reached in hardware by a modified shift register (Dubrova and Mansouri, 2012) and in software by processing the state in units larger than one single bit. Depending on the architecture on which CRC is carried out, the unit can be composed of 4, 8, 16, 32, 64 or 128 bits. The algorithm is speed up by means of a pre-computed lookup table depending only on the coefficients of the CRC generator polynomial.

6 CONCLUSION AND FUTURE WORK

In this paper, we have presented a new lightweight pseudo random number generator and we have shown both its randomness and security. The PRNG is based on very simple operations performed on 1023-bit states, which correspond to increment a state by a suitable odd value *m* and then to reverse the so obtained bit string. Finally, an extended version of CRC is applied, allowing us to produce at each step a 128-bit output number. In this position paper, we have provided a first deep security analysis of our scheme, by showing that it is truly random and resistant to a number of possible attacks. As a future work we plan to deepen the study of the PRNG security against cryptanalysis attacks and to compare our PRNG with existing PRNGs on the aspect of efficiency, also by means of a hardware implementation. Indeed, we guess that our PRNG is very competitive under this point of view, as it is based on very simple operations.

ACKNOWLEDGEMENTS

This work has been partially supported by the TENACE PRIN Project (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research and by the Program “Programma Operativo Nazionale Ricerca e Competitività” 2007-2013, Distretto Tecnologico CyberSecurity funded by the Italian Ministry of Education, University and Research.

REFERENCES

Alcaraz, C. and Lopez, J. (2010). A security analysis for wireless sensor mesh networks in highly critical systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(4):419–428.

Blum, L., Blum, M., and Shub, M. (1986). A simple unpredictable pseudo-random number generator. *SIAM Journal on computing*, 15(2):364–383.

Buccafurri, F. and Lax, G. (2011). Implementing disposable credit card numbers by mobile phones. *Electronic Commerce Research*, 11(3):271–296.

Bundesamt für Sicherheit in der Informationstechnik (2014). http://www.bsi.de/english/index.htm.

Cox, G., Dike, C., and Johnston, D. (2011). Intels Digital Random Number Generator (DRNG). Technical report, Intel.

Dolev, S., Gilboa, N., Kopeetsky, M., Persiano, G., and Spirakis, P. G. (2011). Information security for sensors by overwhelming random sequences and permutations. *Ad Hoc Networks*.

Dubrova, E. and Mansouri, S. S. (2012). A bdd-based approach to constructing lfsrs for parallel crc encoding. In *Multiple-Valued Logic (ISMVL), 2012 42nd IEEE International Symposium on*, pages 128–133. IEEE.

ECMA (1992). *ECMA-182: Data Interchange on 12,7 mm 48-Track Magnetic Tape Cartridges — DLT1 Format*.

EPCglobal, E. (2004). Radio-frequency identity protocols class-1 generation-2 uhf rfid protocol for communications at 860 mhz–960 mhz version 1.0. 9. *K. Chiew et al./On False Authenticationsfor C1G2 Passive RFID Tags*, 65.

Hill, J. R. (1979). A table driven approach to cyclic redundancy check calculations. *SIGCOMM Comput. Commun. Rev.*, 9(2):40–60.

Huang, Y.-J., Yuan, C.-C., Chen, M.-K., Lin, W.-C., and Teng, H.-C. (2010). Hardware implementation of rfid mutual authentication protocol. *Industrial Electronics, IEEE Transactions on*, 57(5):1573–1582.

L’Ecuyer, P. (1994). Uniform random number generation. *Annals of Operations Research*, 53(1):77–120.

Li, Y. and Zhang, X. (2005). Securing credit card transactions with one-time payment scheme. *Electronic Commerce Research and Applications*, 4:413–426. Elsevier Science Publishers B. V.

Melià-Seguí, J., Garcia-Alfaro, J., and Herrera-Joancomartí, J. (2013). J3gen: A prng for low-cost passive rfid. *Sensors*, 13(3):3816–3830.

National Institute of Standards and Technology (2014). Federal Information Processing Standards Publication, Washington.

Rukhin, A., Soto, J., Nechvatal, J., Smid, M., and Barker, E. (2001). A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, DTIC Document.

Schindler, W. (1999). Functionality classes and evaluation methodology for deterministic random number generators. *Federal Office for Information Security (BSI)*.

Tang, B.-y., ZENG, N., ZHENG, L.-x., and CHEN, H.-h. (2004). Design and implementation of web-based remote supervisory system in the embedded system. *Journal-Xiamen University Natural Science*, 43(5):632–635.

Wang, Y. (2011). sSCADA: securing SCADA infrastructure communications. *International Journal of Communication Networks and Distributed Systems*, 6(1):59–78.