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
bucca@unirc.it, lax@unirc.it

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 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 (Melí-Melí 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 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...
cryptography 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 ex-
perimentally in Section 4. In Section 5, we deal with
the computational costs of our PRNG. Finally, in Sec-
tion 6, we draw our conclusions.

2 NOTATION

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

- We denote by $x^j = (x_1, \ldots, x_k)$ 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 $x^k = (x_k, \ldots, x_1)$ the reverse string
  of $x^k$.
- Given a positive integer $p$, $x^k + p$ denotes the $k$-bit
  string representing the number obtained by sum-
  ming (in $2^k$-modulo arithmetic) $x^k$ thought as a bi-
  nary number and $p$. For example, given $p = 1$ and
  $x^3 = 111, x^3 + 1 = 000$, since $(111 + 001) \mod 1000 = 000$.
- 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., $[x^k]_i$.
- We denote by $x^j x^i$ the $(i + j)$-bit string obtained by
  appending $x^i$ to $x^j$.
- Given a $k$-bit string $x$, we denote by $\overline{x}$ 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 $\overline{x} = 0011$.
- Finally, we denote by $1^k$ ($0^k$, resp.) the $k$-bit string
  composed of all 1s ($0$s, 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 $(S, T, O, g, S_0)$
where $S$ is the finite state set, $T : S \rightarrow S$ the tran-
sition function, $O$ the output space, $g : S \rightarrow O$ 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 $s_i$, it is pos-
sible to compute $x_i = O(s_i)$ 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 para-
metric with respect to a positive odd integer $m$ and is
defined as follows.

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

In words, $T(\overline{x^k})$ is obtained by reversing the string
$\overline{x^k}$ 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 require-
ment that the knowledge of $x$ does not give an at-
tacker 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 ef-
fort, we observe that we can be satisfied also by a non-
cryptographic hash function whose inversion is com-
putational feasible, but the number of average collid-
ing 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 profit-
able applied in our case if the knowledge of a col-
diding 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 $s^k_i$ such that $\text{CRC}(s^k_i) = c^w$. 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 $s^k_i$ the k-bit string obtained from $s^k$ by circularly right shifting it as many times as the number of 1s occurring in it).

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

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.

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 $\alpha$ 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 \( * \). 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 \( \circ \) 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 \( \text{CRC}(a \text{ XOR } b) = \text{CRC}(a) \text{ XOR CRC}(b) \), 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., \( x_i = \text{CRC}(s_i) \)) starting from the knowledge of (1) the output of the \( j \)-th state of the user and (2) the XOR between \( s_i \) and \( s_j \). 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 \( s_i \) has both the left-most and the right-most bit 0 (i.e., every four steps), the attacker knows that \( s_i \text{ XOR } s_{i+2} = 10^{k-1}1 \) (recall that, according to our preliminary notations, \( 10^{k-1}1 \) denotes a \( k \)-bit string of the form \( 1 \cdots 1 \), with \( k \) - 2 0s). Thus, the CRC of \( s_{i+2} \) 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 \( s_i \) is of the form 00\( \cdots \cdot \cdot \cdot 01 \), then we expect that the XOR with the state generated two steps ahead is of the form \( 10^{k-1}1 \). Again, if \( s_i \) is of the form 10\( \cdots \cdot \cdot \cdot 00 \), then we expect that the XOR with \( s_{i+2} \) is of the form \( 110^{k-1}1 \). Finally, if \( s_i \) is of the form 10\( \cdots \cdot \cdot \cdot 01 \), then we expect that the XOR with \( s_{i+2} \) 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 \( t < \frac{2^{k+1}}{2^{k-1} + 1} \). Let \( s_i \) be a \( k \)-bit state of the form \( 10^{r-2}e_{i}00 \), where \( e_{i} \) and \( e'_{i} \) are \( j \)-bit strings, \( d^{k-4-2j} \) is a \( (k-4-j) \)-bit string containing at least one 0 and \( j = \lfloor \log_{2}k \rfloor + 1 \). It holds that the sequence \( S' = \langle s_{0}', \ldots, s_{t}' \rangle \) such that \( s_{0}' = T(s') \) and \( s_{k}' = T(s_{t-1}') \) for \( 1 \leq r \leq t \) does not contain any state of the form \( 10^{r-4}01 \), where \( j^{k-4} \) is a \( (k-4) \)-bit string.

**Proof.** The proof is omitted for space reasons. \( \square \)

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 \( \lfloor \log_{2}k \rfloor + 1 \) 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.
we do not generate bad seeds (i.e., seeds of the form 10⋯01). 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 $10s_1s_2s_3s_4s_5s_6s_700$, where $s_1s_2s_3s_4s_5s_6$ and $s_7$ are randomly generated, with the only constraint that $s_7^{\sim 134}$ does not satisfies the above requirement is $\frac{1}{2^7}$. Thus, the blind random generation could be also accepted, because, in our case in which $k = 1023$, this probability is $\frac{1}{2^{1023}}$.

### 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 $\frac{1}{2^{\frac{k}{2}}}$.

**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 $0^k 0$ or $1^k 1$, where $p^2$ 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 $2^{-k+1}$.

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 $2^{-311}$.

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 $T(x^k) = 2^k + m$ is as large as possible, hopefully $2^k$ (i.e., the upper bound). This means that, starting from a $k$-bit string $x^k$, it is possible to generate $2^k - 1$ different new states before re-obtaining $x^k$. This is guaranteed by the next theorem.

**Theorem 4.3.** Given a $k$-bit string $x^k$ with $k \mod 2 \neq 0$, let $S^k$ be the sequence $(s^k_0, \ldots, s^k_{2^k-1})$ such that $s^k_i = T(s^k_{i-1})$ for $1 \leq i \leq 2^k - 1$.

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

**Proof.** We proceed by induction on the length $k$ of the strings.

Basis ($k = 1$). The sequence $S^1$ is $(0, 1)$. 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 $x^k_0$ as the $(k + 2)$-bit string obtained by $0^k 0$. We proceed by computing the next values of $r^k_{k+2}$. By reversing $r^k_{k+2}$ and by adding 1, we have $r^k_{k+2} = T(r^k_0) = (0^k 0) + 1 = 0^k 1$. Similarly, we obtain $r^k_{k+2} = (0^k 1) + 1 = (1^k 0) + 1 = 1^k 1$. Now, after the inversion of $r^k_{k+2}$ we obtain the string $1^k 0 1$, and after the increasing, the last bit of the string becomes 0, with a carry bit to $r^k_0$. Thus, $r^k_{k+2} = (1^k 0 1) 0 = 0^� 1 0$. Again, we obtain that $r^k_{k+2} = (0^k 1) + 1 = 0^k 1 0 0 0 = 0^k 0$, since $r^k_{k+2} = T(r^k_0) = r^k_1 + 1$. We can easily generalize the above reasoning to the first $2^{k+1}$ elements of $R^{k+2}$. In particular, for each $r^k_{k+2}$ with $0 \leq i < 2^{k+1}$, we have that:

1. if $i \mod 4 = 0$, then $r^k_{k+2} = 0^k 0$
2. if $i \mod 4 = 1$, then $r^k_{k+2} = 0^k 1$
3. if $i \mod 4 = 2$, then $r^k_{k+2} = 1^k 1$
4. if $i \mod 4 = 3$, then $r^k_{k+2} = 1^k 0$

where $j = (i/4) \ast 2$ and $/ \ast /$ 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 $r^k_{k+2} = 1^k 0$. We prove Claim 1 by induction on $k$. The statement is thus proved. The theorem follows immediately from Claim 1.
(3) $2^k - i < a < 2^k$. (1) By construction, $r^k_a = s^k_{i+j}$ for $0 \leq a < 2^k - i$. As a consequence, $r^k_{2^k-i-1} = s^k_{2^k-i-1} = 1$. (2) Since $T(r^k_{2^k-i-1}) = T(1^k) = 0^k$ we have that $r^k_{2^k-j} = s^k_{0^j}$. (3) By construction, $r^k_a = s^k_{i+j}$ 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) $S^i = \langle s^i_0, \ldots, s^i_{2^k-1} \rangle$ such that $s^i_k = T(s^i_{k-1})$ for $1 \leq i \leq 2^k - 1$, let $S^8 = \langle s^8_0, \ldots, s^8_{2^8-1} \rangle$.

Then it holds that $s^8_i \neq s^8_j$, for each $i$ and $j$ such that $0 \leq i < j \leq 2^k - 1$.

**Proof.** We proceed by contradiction. Suppose that $s^8_i \rightarrow s^8_j$ with $i \neq j$. Let $u$ be the number of 1s in $s^8_i$ (and, consequently, also in $s^8_j$). Now, shifting both numbers by $u$ left circular shifts, we obtain $s^8_i$ and $s^8_j$, respectively, with $s^8_i = s^8_j$ by construction. Since this contradicts Theorem 4.1, it results that $s^8_i \neq s^8_j$.

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 $g(s_i) = CRC128(\overline{s_0^i}) = PRN_i$. Since the number of colliding states w.r.t. $CRC128$ 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 $\overline{s_0^i}$ chosen by the attacker (among the $2^{895}$ states) differs from the actual $s_i$ (i.e., the current state), then the probability that $g(T(\overline{s})) = g(T(s_i))$ is $\frac{1}{2^{895}}$. 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 consecutive output numbers. By a brute force attack, the attacker should test $(2^{128})^c$ 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

$$\sum_{i=1}^{m} \alpha^i \leq 2.$$
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 TRENCE 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


Elsevier Science Publishers B. V.


