Warbler: A Lightweight Pseudorandom Number Generator for EPC C1 Gen2 Passive RFID Tags

Kalikinkar Mandal, Xinxin Fan and Guang Gong
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1

Abstract

A pseudorandom number generator is an important component for implementing security functionalities on RFID tags. Most previous proposals focus on true random number generators that are usually inefficient for low-cost tags in terms of power consumption, area, and throughput. In this contribution, we propose a lightweight pseudorandom number generator (PRNG) for EPC Class-1 Generation-2 (EPC C1 Gen2) RFID tags. The proposed PRNG fully exploits nonlinear feedback shift registers and provides 16-bit random numbers that are required in the tag identification protocol of the EPC C1 Gen2 standard. The generated sequences are able to pass the EPC C1 Gen2 standard's statistical tests as well as the NIST randomness test suite. Moreover, a detailed cryptanalysis shows that the proposed PRNG is resistant to the most common attacks such as algebraic attacks, cube attacks, and time-memory-data tradeoff attacks. In particular, the proposed PRNG can be implemented on low-cost Xilinx Spartan-3 FPGA devices with 46 slices.

1. Introduction

Radio Frequency Identification (RFID) is a promising technology for automatic identification of remote objects. In an RFID system, each object is labeled with a small transponder, called an RFID *tag*, which receives and responds to radio-frequency queries from a transceiver, called an RFID *reader*. An RFID tag is composed of a tiny integrated circuit for storing and processing identification information, as well as a radio antenna for wireless data transmission. There are three basic types of RFID tags: *active*, *semi-passive*, and *passive* tags. Active tags contain internal batteries so that they can initialize communications with the reader, whereas a passive tag does not contain any battery, it solely obtains power from the reader for both computation and communication. Semi-passive tags use batteries only to power their circuit and harvest power from the reader for communication. Passive RFID tags usually have constrained capabilities in every aspect of computation, communication and storage due to the extremely low production cost. The reading range of a passive tag is up to several meters. For most RFID applications, the security and privacy are important or even crucial requirements [16]. Since most protocols for securing RFID systems proposed so far are based on the usage of an on-board true random and/or pseudorandom number generator (TRNG/PRNG), a number of solutions have been proposed in the literature for implementing TRNGs/PRNGs on RFID tags [2, 7, 15, 19, 20, 22]. In particular, the EPCglobal Class-1 Generation-2 (EPC C1 Gen2 in brief) standard [10] uses a couple of 16-bit random numbers in the tag identification protocol. All of the proposals for TRNGs are based on analog circuits that sample a random physical phenomenon like thermal noise. To the best of our knowledge, only three PRNGs have been proposed for EPC C1 Gen2 passive tags [7, 20, 22], among which two proposals use TRNGs as a component and the security properties of those two PRNGs rely on the security of TRNGs.

Considering the high power consumption, large area and low throughput of TRNGs, we propose a lightweight PRNG for low-cost EPC C1 Gen2 tags in this paper. The basic idea of our design is to replace the TRNG in [7, 20] by a lightweight pseudorandom sequence generator with good statistical properties. To this end, nonlinear feedback shift registers (NFLSRs) have been fully exploited in our design. The security properties of the proposed PRNG are analyzed in great detail by using cryptographic statistical tests specified by the EPC C1 Gen2 standard as well as the NIST test suite. Various cryptanalysis techniques have been applied to demonstrate the attack resistant properties of the proposed PRNG. Furthermore, a hardware implementation on a Xilinx Spartan-3 FPGA device shows that the new PRNG can be implemented using 46 slices.

1.1. EPC C1 Gen2 Protocol

The EPC C1 Gen2 was approved as ISO 18000-6C standard in 2006 for global use. Figure 1 shows an overview of the tag identification protocol. In the EPC C1 Gen2 tag identification protocol, two main operations, namely *inventory* and *access*, are performed for managing the tag population. In the inventory operation (Steps 1-4 in Figure 1), after

receiving a request from the reader, a tag generates a 16-bit random number, denoted by *RN16*, and temporarily stores the number in a slot counter. When the slot counter is zero, the tag backscatters *RN16* to the reader. Thereafter, the reader copies *RN16* to an *acknowledgement* packet to be sent to the tag. When the tag receives the acknowledgement packet, it first compares the random number in the acknowledgement packet with *RN16*. If these two numbers are the same, then the tag backscatters the acknowledgement packet.

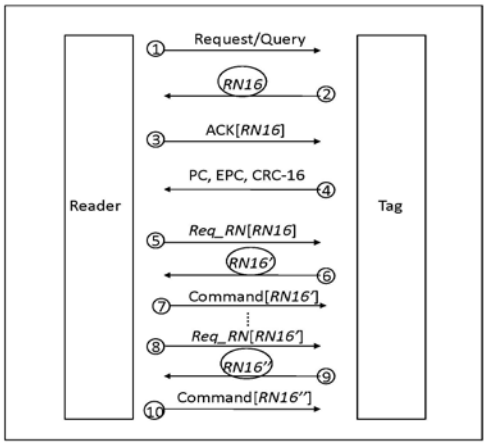

Figure 1. EPC C1 Gen 2 Tag Identification Protocol

In the access operation (Steps 5-7 in Figure 1), after receiving a request, denoted by Req_{RN}, from the reader, the tag compares the random number in the request Req_{RN} with the stored *RN16*. If these two random numbers match, then the tag generates another random number *RN16'*, which is called *handle* and backscatters it to the reader. Then the reader issues the commands such as *Read*, *Write*, and *BlockWrite*. Steps 8-10 in Figure 1 demonstrate a further access operation. Note that for each access operation the tag generates a new random number.

2. Related Work

In this section, we give a brief overview of three previous PRNG proposals for EPC C1 Gen-2 passive RFID tags.

2.1. Che *et al.*'s PRNG

Che *et al.* [7] designed a PRNG based on a combination of an oscillator-based TRNG and a linear feedback shift register (LFSR) with 16 stages. In their design, the TRNG is implemented using an analog circuit and exploits thermal noise of the circuit. To introduce randomness, one truly random bit from the TRNG is XORed with each bit of a 16-bit sequence generated from the LFSR. In 16 clock cycles, a 16-bit random number is generated by the PRNG. Due to the linear structure, Che *et al.*'s scheme has been attacked by Melia-Segui *et al.* in [20] with a high success probability $\frac{n+1}{8n}$, where *n* is the length of the LFSR.

2.2. Melia-Segui *et al.*'s PRNG

To avoid such an attack on Che *et al.*'s PRNG, Melia-Segui *et al.* [20] proposed a similar design by employing multiple primitive polynomials instead of one in the LFSR. The design consists of a true random source, a module with eight primitive polynomials, and a decoding circuit taking inputs from the true random source, where the decoding circuit is designed in such a way that the same primitive polynomial is not chosen consecutively. At each clock cycle, one primitive polynomial is chosen according to the decoding logic and true random bits for producing a pseudorandom bit. Thus, the PRNG produces a 16-bit random number in 16 clock cycles and the security of the PRNG relies on the TRNG.

2.3. Peris-Lopez *et al.*'s PRNG

In [22], Peris-Lopez *et al.* proposed a PRNG named LAMED for RFID tags, which is in compliance with the EPC C1 Gen2 standard and can provide 32-bit as well as 16-bit random numbers. The basic operations for updating the internal state of LAMED consist of bitwise XOR operations, modular algebra, and bit rotations. The internal state of the LAMED is of 64-bit, including a 32-bit key and a 32-bit initial vector. The key length can be further increased by replacing the IV bits with the key bits. Note that the LAMED always outputs a 32-bit random number and a 16-bit random number is obtained by dividing 32-bit number into two equal halves and XORing them together.

3. Preliminaries

In this section, we define some terms and notations that will be used to describe the proposed pseudorandom number generator.

Notations:

- $F_2 = GF(2) = \{0,1\}$: the Galois field with two elements.
- $p(x) = 1 + x + x^3 + x^4 + x^5$: a primitive polynomial over F_2.
- $F_{2^5} = GF(2^5)$: the extension field of F_2 with 2^5 elements. Let α be a primitive element of F_{2^5} such that $p(\alpha) = 0$.
- $Tr(x) = x + x^2 + x^{2^2} + x^{2^3} + x^{2^4}$: the trace function from F_{2^5} to F_2.

Definition 1. The *linear span* (LS) or *linear complexity* of a binary sequence is defined as the

length of the smallest linear feedback shift register which generates the entire binary sequence.

Definition 2. A binary sequence with period $2^n - 1$ is called a *span n* or *modified de Bruijn sequence* if each non-zero *n*-tuple occurs exactly once in a period.

Definition 3. Two periodic sequences of the same period are called shift distinct of each other if one sequence cannot be obtained from the shift equivalent of the other sequence.

Definition 4. An imbalance range of a binary sequence is the absolute difference between the number of zeros and ones in a period.

The WG Transformation: Let m mod $3 \neq 0$, $3k$ mod $m = 1$ and $h(x) = x + x^{q_1} + x^{q_2} + x^{q_3} + x^{q_4}$ where q_i's are given by $q_1 = 2^k + 1, q_2 = 2^{2k} + 2^k + 1, q_3 = 2^{2k} - 2^k + 1, q_4 = 2^{2k} + 2^k - 1$. Then the function WGP(), mapping from F_{2^m} to F_{2^m}, given by

$$\mathrm{WGP}(x) = h(x+1) + 1$$

is called the *WG permutation* and the function f from F_{2^m} to F_2 given by

$$f(x^d) = Tr\big(\mathrm{WGP}(x^d)\big),\ x \in F_{2^m}$$

is known as the *WG transformation* with decimation d, where d is co-prime to $2^m - 1$ [12]. The WG transformation has excellent cryptographic properties such as high nonlinearity, algebraic degree and at least 1-order resiliency for a proper selection of basis. Moreover, a sequence generated by the WG transformation has high linear complexity.

4. Description of the Proposed PRNG

The proposed PRNG is composed of two main building blocks. The first one consists of two NLFSRs of length 17 and 18 over F_2, each one generating a span *n* sequence or modified de Bruijn sequence with optimal linear complexity, whereas the second one includes an NLFSR over F_{2^5} and each NLFSR uses one or two WG transformation modules. In our design, the binary sequence generated by the first building block is converted to a sequence over F_{2^5} and this sequence is used in the recurrence relation in the second building block. The final output sequence is filtered by the WG transformation and *n*-bit random numbers are generated by taking disjoint *n*-bit sequences from the final output sequence. A high-level architecture of the proposed PRNG is illustrated in Figure 2.

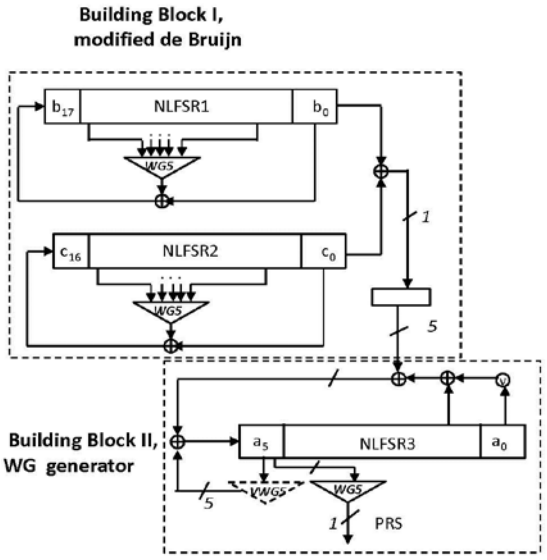

Figure 2. A Diagram of the PRNG for EPC C1 Gen2 Tags

4.1. Building Block I: An Alternative to TRNG

The first building block contains two NLFSRs whose lengths are chosen to be co-prime in order to achieve the maximum period. The reason that two shorter NLFSRs are used instead of a long one is due to the impossibility of generating shift distinct sequences from a long NLFSR for different initial states. In other words, by XORing the output sequences from two NLFSRs we can obtain shift distinct sequences for different initial states. In our design, the WG transformation with decimation $d = 3$ over F_{2^5}, denoted by *WG5* in Figure 2, is used as a nonlinear feedback function to generate span *n* sequences. For $m = 5$, the WG permutation is defined as

$$WGP_5(x) = x + (x+1)^5 + (x+1)^{13} + (x+1)^{19} + (x+1)^{21}, x \in F_{2^5}$$

and the WG transformation over F_{2^5} is given by

$$f(x) = Tr\big(WGP_5(x)\big) = Tr(x^{19}), x \in F_{2^5},$$

which has the maximum nonlinearity 12, the algebraic degree 3 and the maximum algebraic immunity 3. The *n*-stage nonlinear recurrence relation is defined as

$$b_{n+k} = b_k \oplus f(x^d), x = \big(b_{r_{1+k}}, \dots, b_{r_{5+k}}\big) \in F_{2^5}$$

for all $k \geq 0$, and $0 < r_1 < \cdots < r_5 < n$ are tap positions of two NLFSRs, where $\oplus$ denotes addition over F_2. Using the parameters and recurrence relations in Table 1, we can generate two span *n* sequences $\boldsymbol{b} = \{b_i\}_{i\geq 0}$, and $\boldsymbol{c} = \{c_i\}_{i\geq 0}$ with NLFSR1 and NLFSR2, respectively. The output sequence of the first building block is denoted by $\boldsymbol{s} = \{s_i\} = \{b_i \oplus c_i\}$, $i \geq 0$ which is almost balanced and has the following statistical properties:

a) The period is $(2^{18} - 1)(2^{17} - 1) \approx 2^{35}$;

b) The imbalance range is 4; and
c) The linear span is $(2^{18} + 2^{17} - 4) \approx 2^{18.58}$.

For different initial states of the NLFSRs, the number of shift distinct sequences $\boldsymbol{s}$ is equal to $(2^{18} - 1)(2^{17} - 1) - 2$.

Table 1. Parameters and Statistical Properties of Two NLFSRs

NLFSR	Length	Tap position $(r_1 \ldots r_5)$	Period	Linear Span
NLFSR1	18	4, 7, 8, 10, 15	$2^{18} - 1$	$2^{18} - 2$
NLFSR2	17	4, 7, 8, 9, 12	$2^{17} - 1$	$2^{17} - 2$

We now generate a new sequence $\boldsymbol{t} = \{t_k\}_{i \geq 0}$ over F_{2^5} from $\boldsymbol{s}$ as follows

$$t_k = (s_{5k}, s_{5k+1}, \ldots, s_{5k+4}) \in F_{2^5}, k \geq 0.$$

Note that the sequence $\boldsymbol{t}$ is a shift distinct sequence for different initial states of the NLFSRs and the linear complexity of sequence $\boldsymbol{t}$ is bounded below by $2^{18.58}$ as F_{2^5} and F_2 have the same characteristic [17]. The sequence $\boldsymbol{t}$ is used in the second building block for introducing nonlinearity in the recurrence relation in each 5 clock cycles (see Section 5 for details). This building block is used as an alternative to the TRNG in [7, 20].

4.2. Building Block II: Pseudorandom Number Generator

The second building block consists of an NLFSR and two WG transformation modules given by $f(x)$ and $f(x^3)$, respectively. Letting the length of NLFSR3 be $l = 6$ and the primitive polynomial be $g(x) = x^6 + x + \gamma$, where $\gamma = \alpha^{15} \in F_{2^5}$, the recurrence relation is defined as

$$a_{k+6} = \gamma a_k + a_{k+1} + w_k + t_k, a_i \in F_{2^5}, k \geq 0 \quad (1)$$

where $w_k = (0, 0, 0, 0, f(a_{k+5}))$ is the nonlinear feedback with the least signification bit generated by the WG transformation $f(x)$ and $\boldsymbol{t} = \{t_k\}_{i \geq 0}$ is the sequence over F_{2^5} that is defined in the previous subsection. While the WG transformation $f(x)$ (i.e., VWG5 in Figure 1) is only used as a nonlinear feedback function in NLFSR3, the WG transformation $f(x^3)$ (i.e., WG5 in Figure 1) is employed to generate nonlinear feedback for NLFSR1 and NLFSR2 as well as to filter the output sequences. In the above recurrence relation (1), the nonlinearity is introduced by t_k and w_k and those feedback will affect other bit positions after multiplying by γ. Note that the period of the sequence $\boldsymbol{a} = \{a_k\}_{i \geq 0}$ is a multiple of the period of $\boldsymbol{t}$. Moreover, the final output sequence $\boldsymbol{o} = \{o_k\}$ of the second building block is defined by $o_k = f(a_{k+5}^3)$ for $k \geq 0$. The period of $\boldsymbol{o}$ is a multiple of $\frac{2^{35}}{5} = 2^{32.67}$, and the linear complexity of $\boldsymbol{o}$ is lower bounded by the linear complexity of $\boldsymbol{t}$.

4.3. System Initialization

The proposed PRNG has an internal state 65 bits, including a 45-bit secret seed as well as a 20-bit initial vector (IV). While the secret seed and the IV are preloaded into RFID tags at the very beginning, the 20-bit IV is also updated at the end of each protocol session. Before generating random numbers, a 36 rounds of initialization phase is applied to mix the key and IV properly. In our design, the secret seed and IV are preloaded as follows: the first consecutive 12, 11 and 22 positions of NLFSR1, NLFSR2 and NLFSR3 are respectively reserved for key bits, whereas the remaining positions in each NLFSR are for the IV. The initialization process is illustrated in Figure 3. During the initialization phase the internal states of the three NLFSRs are updated as follows:

$b_{18+k} = b_k \oplus f(x^3) \oplus o_k,$
$x = (b_{k+4}, b_{k+7}, b_{k+8}, b_{k+10}, b_{k+15}), k \geq 0, o_0 = 0,$
$c_{17+k} = c_k \oplus f(y^3) \oplus o_k,$
$y = (c_{k+4}, c_{k+7}, c_{k+8}, c_{k+9}, c_{k+12}), k \geq 0, o_0 = 0,$
$s_{k+4} = b_k \oplus c_k, k \geq 0, s_j = 0, j = 0, 1, 2, 3,$
$t_k = (s_k, s_{k+1}, s_{k+2}, s_{k+3}\, s_{k+4}) \in F_{2^5}, k \geq 0,$
$a_{k+6} = \gamma a_k + a_{k+1} + w_k + t_k, a_i \in F_{2^5}, k \geq 0,$
$w_k = \big(0, 0, 0, 0, f(a_{k+5})\big), k \geq 0,$
$o_{k+1} = f(a_{k+5}^3), k \geq 0,$

where b_{18+k}, c_{17+k} and a_{k+6} are the updated values of NLFSR1, NLFSR2 and NLFSR3, respectively, and w_k is generated by the WG transformation $f(x)$. Sequence $\{s_k\}$ is the XOR sequence of two output bits from NLFSR1 and NLFSR2 and five consecutive s_k's are collected to form a 5-bit vector t_k. The output o_k of NLFSR3 is used as a nonlinear feedback to affect the internal states of both NLFSR1 and NLFSR2.

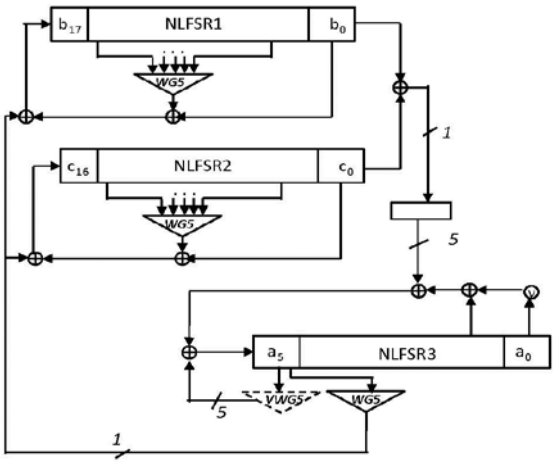

Figure 3. Key Initialization Phase

Remark 1. A 20-bit IV can be generated from the initial SRAM state of tags when tags are powered up (see [15]). The entropy of IV can also be increased by employing the von Neumann technique, which can be efficiently implemented in hardware [24]. However, the implementation of these components needs additional hardware support.

5. Security Analysis of the PRNG

The security analysis of the proposed PRNG is conducted in two steps. In the first step, we performed all cryptographic statistical tests that are specified in the EPC C1 Gen2 standard [10] and the NIST standard [23] on several sets of pseudorandom sequences generated by the proposed PRNG with different initial states. In the second step, we investigate the attack resistant properties of the new PRNG by launching the algebraic attacks, cube attacks, and time-memory-data tradeoff attacks.

5.1. Randomness Analysis of the PRNG

According to the EPC C1 Gen2 standard, a true random or pseudorandom number generator must satisfy the following three statistical properties:

- **Probability of a single sequence:** The probability that any 16-bit random sequence (*RN16*) drawn from the PRNG has value *j*, shall be bounded by
  $\frac{0.8}{2^{16}} \leq \Pr(RN16 = j) \leq \frac{1.25}{2^{16}}$ for any *j*.
- **Probability of simultaneously identical sequences:** For a tag population up to 10,000, the probability that any of two or more tags simultaneously generate the same sequence of bits shall be less than 0.1%, regardless of when the tags are energized.
- **Probability of predicting a sequence:** A sequence drawn from the PRNG 10ms after the end of transmission shall not be predictable with a probability greater than 0.025% if the outcomes of prior draws from the PRNG, performed under identical conditions, are known.

We implemented our PRNG in software for checking whether the proposed PRNG meets the above three criteria. To verify the first criterion, we generated 18 different test sequences for different initial states of the NLFSRs and calculated the probability of occurrence of 16-bit numbers. Our experimental results show that the probability of any 16-bit number *j*, i.e., $\Pr(RN16 = j)$ lies between $\frac{0.9409}{2^{16}}$ and $\frac{1.0693}{2^{16}}$, which are better bounds than those obtained in [20]. The upper and lower bounds of probability values for different tests are given in 2nd and 3rd columns in Table 2. With respect to the second criterion, our PRNG can generate up to $2^{45} - 1$ shift distinct sequences for different keys to each tag, since the sequence ***t*** generated in Section 4.1 is shift distinct. Thus the probability that any two tags will generate the same sequence with period at least $2^{32.67}$ is approximately 2^{-45} that is much less than 0.1%. For the third criterion, given a 16-bit random number, an attacker can recover the internal state of NLFSR3 with probability 2^{-24} after getting 80 bits of the sequence ***s***. To obtain the next 16-bit random number from the given one, the adversary needs to know the next consecutive 80 bits of the sequence ***s*** and the internal state of NLFSR3. The 80 bits can be obtained either by guessing or obtaining about $\frac{2^{18.58}}{5} = 2^{16.26}$ consecutive random numbers. Due to the high linear span of the sequence ***s***, it is impossible to generate the next consecutive 80 bits from previous known 80 bits in practice. Furthermore, it is also difficult for an adversary to intercept $2^{16.26}$ consecutive random numbers in one protocol session because the communication session in RFID systems is usually quite short and the IV is different. Moreover, the secret seed can also be updated for different sessions. Hence, the attacker can guess the next 16-bit random number with the better probability 2^{-16}, which is much less than 0.025% as specified in the EPC C1 Gen2 standard.

To measure the linear dependency between an *n*-bit output and the previous *n*-bit output, we performed a serial correlation test [18] on the sequences generated by the PRNG. We generated 18 distinct sequences for different initial values of the NLFSRs, each one is of size 2^{26} bytes and calculated the serial correlation coefficient for 1-bit, 1-byte and 2-byte lag. Our experimental results demonstrate that the serial correlation coefficients are close to zero, which indicates the good randomness of the generated sequences. The serial correlation coefficients for different sequences are given in 4th, 5th and 6th columns of Table 2.

Table 2. The First and Third Requirements

Seq.	Upper	Lower	1-bit	1-byte	2-byte
S1	1.0471	0.9497	0.000098	-0.000080	-0.000061
S2	1.0476	0.9530	-0.000012	0.0000025	-0.000055
S3	1.0444	0.9555	0.000094	-0.000064	-0.000006
S4	1.0693	0.9517	-0.000075	0.000106	-0.000046
S5	1.0468	0.9537	0.000057	0.000041	-0.000041
S6	1.0440	0.9545	-0.000012	0.000012	0.000078
S7	1.0457	0.9550	-0.000063	-0.000028	0.000080
S8	1.0454	0.9560	0.000025	0.000085	0.000032
S9	1.0533	0.9550	-0.000002	-0.000005	-0.000042
S10	1.0483	0.9544	0.000082	-0.000023	0.000023
S11	1.0541	0.9532	0.000045	-0.000033	0.000046
S12	1.0456	0.9514	0.000030	0.000026	0.0000012
S13	1.0487	0.9493	-0.000006	0.000101	0.000071

S14	1.0494	0.9523	-0.000053	-0.000047	0.000036
S15	1.0506	0.9550	-0.000075	-0.000091	-0.000086
S16	1.0302	0.9850	0.000015	0.000004	-0.000106
S17	1.0499	0.9505	-0.000091	0.000025	-0.000067
S18	1.0533	0.9409	0.000012	-0.000028	-0.000043

Different from the statistical tests in the EPC C1 Gen2 standard, the NIST test suite contains 15 demanding statistical tests for characterizing the randomness of a binary sequence. According to the NIST specification [23], a PRNG passes the test suite successfully if it passes all the tests simultaneously with a proportion of 96%. In our experiment, 10 test sequence (TS) sets are generated, each of which has 100 different sequences with different initial values and each sequence has a length of 2^{25}. We computed the proportion values for each TS set and listed the test results for 5 TS sets in Table 3. Non-overlapping template matching test results are not given in Table 3 because of 148 entries. However, the proposed PRNG has passed the test successfully. It is not difficult to find out that each TS set can pass the NIST test suite successfully.

Table 3. NIST Test Suite Results of our Proposal

Tests	TS1 Proportion	TS2 Proportion	TS3 Proportion	TS4 Proportion	TS5 Proportion
Frequency	0.97	1.00	0.99	0.98	1.00
Block-frequency	0.99	1.00	0.98	0.99	1.00
Cumulative-sum	0.97, 1.00	1.00, 1.00	0.97, 0.97	0.99, 0.99	0.99, 1.00
Runs	1.00	0.98	1.00	0.99	1.00
Longest-run	0.98	1.00	0.98	0.99	0.98
Rank	0.99	1.00	0.99	1.00	0.99
DFT	1.00	1.00	0.98	1.00	0.99
Overlapping-templates	0.96	0.97	0.97	0.97	0.99
Universal-statistics	0.99	0.98	1.00	1.00	0.98
Approximate entropy	0.99	1.00	0.98	0.97	0.99
Serial	0.99, 0.98	0.98, 0.98	1.00, 1.00	1.00, 1.00	0.99, 1.00
Linear-complexity	0.99	0.99	0.98	0.99	0.99
Random-excursions	0.97, 0.90	0.98, 1.00	0.98, 1.00	1.00, 0.99	0.99, 0.97
	0.97, 0.97	0.98, 0.97	1.00, 0.99	1.00, 0.98	0.98, 0.97
	0.98, 1.00	0.97, 0.97	1.00, 0.99	0.98, 0.97	0.99, 1.00
	0.97, 0.96	0.98, 0.97	0.98, 0.97	0.99, 0.98	1.00, 0.99
Random-excur- variant	0.98,0.98, 0.98	1.00,1.00, 1.00	1.00,1.00, 1.00	0.99,0.98, 0.99	0.98,0.97, 0.99
	0.98,0.98, 0.98	1.00,0.97, 1.00	1.00,1.00, 0.99	1.00,1.00, 1.00	1.00,1.00, 0.99
	1.00,1.00, 0.99	1.00,0.98, 0.98	1.00,1.00, 1.00	1.00,1.00, 1.00	0.99,1.00, 0.99
	1.00,1.00, 1.00	0.98,0.98, 0.98	1.00,1.00, 1.00	0.99,1.00, 0.99	0.99,0.99, 1.00
	0.98,0.98, 0.98	0.98,0.96, 0.96	1.00,1.00, 1.00	0.97,0.98, 1.00	1.00,0.98, 1.00
	1.00,1.00, 1.00	0.98,0.98, 0.98	1.00,0.99, 0.99	0.97,0.96, 0.96	1.00,0.99, 0.98

5.2. Cryptanalysis of the PRNG

In this subsection, the attack resistant properties of the PRNG are investigated by considering the algebraic attacks, cube attacks, and time-memory-data tradeoff attacks in detail. Since our PRNG uses nonlinear feedback shift registers over different fields, we also explain below why the correlation attacks [21], Discrete Fourier Transformation (DFT) attacks [13], and differential attacks [25] are not applicable.

5.2.1. Algebraic Attack

Algebraic attack [8] is a powerful attack against stream ciphers. In our PRNG design, nonlinear feedback functions are used to update the internal states of different NLFSRs and the output bits are filtered by the WG transformation. Noting that the length of the internal state of the PRNG is 65-bit and the length of the secret key is 45-bit, one can reduce the PRNG to a system of linear equations with about 2^{45} unknown variables after applying the initialization round, which can be solved by approximately $\frac{7}{64}(2^{45})^{log_2(7)}$ operations. As a result, the algebraic attack is not better than the exhaustive search in this case.

5.2.2. Cube Attack

Cube attack [9] is a generic key-recovery attack that can be applied to any cryptosystem, provided that the attacker can obtain a bit of information that can be represented by a low-degree decomposition multivariate polynomial in Algebraic Normal Form of the secret and public variables of the target cryptosystem. According to the cube attack, our PRNG can be regarded as a system of multivariate polynomials $p(k_1, \dots, k_{45}, v_1, \dots, v_{20})$ with public IV variables $v_1, v_2 \dots, v_{20}$ and secret key variables $k_1, k_2, \dots, k_{45}$. The polynomial $p(k_1, \dots, k_{45}, v_1, \dots, v_{20}) = t_I . p_{S(I)} + q(k_1, \dots, k_{45}, v_1, \dots, v_{20})$ is called a *master* polynomial, where $t_I = v_{i_1} v_{i_2} \dots v_{i_k}$ is a monomial with $\{i_1, i_2, \dots, i_k\} \subseteq \{1, 2, \dots, 20\}$ and $p_{S(I)}$ is called a *superpoly* of t_I in p. The term t_I is called a *maxterm* if $deg(p_{S(I)}) = 1$. We implemented the cube attack against our PRNG in CUDA and exploited the power of a GPU (i.e, a Tesla C2070 from NVIDIA) for accelerating the computation significantly. We took the first output bit after the 36-round initialization phase in order to find the maxterms in the master polynomial and performed an exhaustive search over all possible cube dimensions ranging from 1 to 20. However, our experiment did not find any linear and quadratic superpoly equations for different cube dimensions.

5.2.3. Time-Memory-Data Tradeoff Attack

Time-memory-data tradeoff attack is a generic cryptanalytic attack which can be applied to any cipher. In a stream cipher, the complexity of a time-memory-data tradeoff attack depends on the length of the internal state, which is given by $O(2^{\frac{n}{2}})$, where n is the length of the internal state [4]. We note that a stream cipher with low sampling resistance is vulnerable to a more flexible time-memory-data tradeoff attack. In our PRNG, the WG transformation is the filtering function as well as the internal state update function and the number of terms in the algebraic normal form representation of the WG transformation is 15, among which only two terms are linear and the remaining terms are either quadratic or cubic. Only by fixing four input variables in the WG transformation, one can obtain a linear function in one variable. Thus, the sampling resistance of the proposed PRNG is high. Since the length of the internal state is 65-bit in our PRNG, the expected complexity of the time-memory-data tradeoffs attack is $O(2^l)$, where l equals 32.5.

5.2.4. Other Attacks

In the fast correlation attacks [21], the internal state of an LFSR based stream cipher can be recovered by first determining a system of linear equations according to a statistical model and then solving the system of linear equations. In our PRNG, the internal state is updated in a nonlinear way. Thus it is hard for an attacker to decide such a system of (non-) linear equations according to some statistical models.

For an LFSR based stream cipher, the DFT attacks [13] can be applied when the exact linear complexity of the output sequence and enough consecutive output bits are known. In our PRNG, the exact linear complexity and period of the output sequence are not known for an initial state. Therefore, the DFT attacks cannot be applied to our PRNG. Moreover, in the EPC C1 Gen2 standard protocol, it is hard for an attacker to obtain enough consecutive bits.

A chosen IV attack on the original version of WG cipher was presented in [25], where one can distinguish several bits of the output sequence by building a distinguisher based on differential cryptanalysis. In our PRNG, two nonlinear terms (i.e., an output from the WG transformation as well as a 5-bit tuple generated by the first building block) are added to the recurrence relation. Thus the differentials after 36 rounds of the initialization phase will contain most internal state bits. As a result, it would be hard for an attacker to distinguish output bits generated by the proposed PRNG.

5.3. Comparisons with Sponge-based PRNGs

A sponge-based PRNG is constructed using a sponge function [3], which is composed of two phases: an *absorbing* phase and a *squeezing* phase. While truly random seeds are fed into the internal state of the sponge function in the absorbing phase, the squeezing phase outputs pseudorandom numbers [3]. Any sponge-based hash function such as U-QUARK [1], DM-PRESENT [6], PHOTON [14], and SPONGENT [5] can be used to construct a sponge-based PRNG. Since a sponge-based PRNG requires a multiple seeding mechanism, one needs additional random sources to provide multiple truly random seeds to the PRNG while generating pseudorandom numbers. Note that the multiple seeding mechanism provides the forward secrecy for the sponge-based PRNGs, which can also be achieved by our PRNG provided that additional random sources are available[1]. Moreover, our PRNG

[1] Assume that the length of a seed K is a multiple of the length of the secret key $K = k_1 || k_2 || \dots || k_r, r \geq 1$. If a multiple seeding mechanism is applied to our PRNG, the seed is updated by repeating the following step r times: k_i $(i = 1, 2, \dots, r)$ is XORed with the values in the key bit positions of the current internal state, followed by an 18-round of the initialization phase.

design has desired randomness properties like period and linear span (see Table 4).

6. Hardware Implementation

In this section we report efficient FPGA implementation of the proposed PRNG core on the low-cost Xilinx FPGA series Spartan-3 and compare our results with other reported lightweight PRNG implementations.

6.1. Target Platform and Design Tools

FPGAs are composed of configurable logic blocks (CLB) and a programmable interconnection network. We implement the proposed PRNG core in VHDL for the low-cost Spartan-3XC3S50 (Package PQ208 with speed grade -5) FPGA device from Xilinx [26]. Considering the availability of SRL16 in Spartan-3 generation of FPGAs, we have coded NLFSR1, NLFSR2, and NLFSR3 properly to guide the synthesis tool to infer SRL16 shift register cells, which enables us to reduce the area of the resulting implementation significantly. We use the integrated FPGA development environment Aldec Active-HDL 9.1 for writing, debugging and simulating VHDL codes. Furthermore, Synopsys Synplify Premier with Design Planner E-201103-SP2 and Xilinx ISE Design Suite v13.2 are employed for the design synthesis and implementation, respectively.

The PRNG core works as follows. Under the control of clock enable pins CE1, CE2, and CE3, the 45-bit secret seed and 20-bit IV will be first loaded into NLFSR1, NLFSR2, and NLFSR3 within 18 clock cycles through pins DIN1, DIN2, and DIN3, respectively. After loading the required key and IV, the initialization phase will be performed in the next 36 clock cycles without any output (i.e., CE5 is disabled in this phase). The running phase will start from the 55-th clock cycle and the PRNG core will output one bit every five clock cycles under the control of clock enable pins CE3 and CE5.

The finite state machine (FSM) has two 1-bit input signals CLK and RST and seven 1-bit output control signals CE1-- CE5, Init, and Load. In our design, a binary counter is used to keep track of the number of clock cycles elapsed. The FSM starts by pulling up the reset signal RST to '1', which resets the counter to be 0. At this time instance, the FSM sets two control signals Init = '0' and Load = '1' and starts loading key and IV. When the counter reaches a values of 17, the FSM goes into the initialization phase and two control signals become Init = '1' and Load = '0', respectively. During the initialization phase, the counter continues increasing by one at every clock cycle until it hits a value of 53. The FSM then transfers to the running phase. During the running phase, both control signals *Init* and *Load* are set to be '0' and a 16-bit random number will be generated every 16 clock cycles.

6.2. Hardware Architecture

A top-level description as well as the data path of the proposed PRNG core is illustrated in Figure 3.

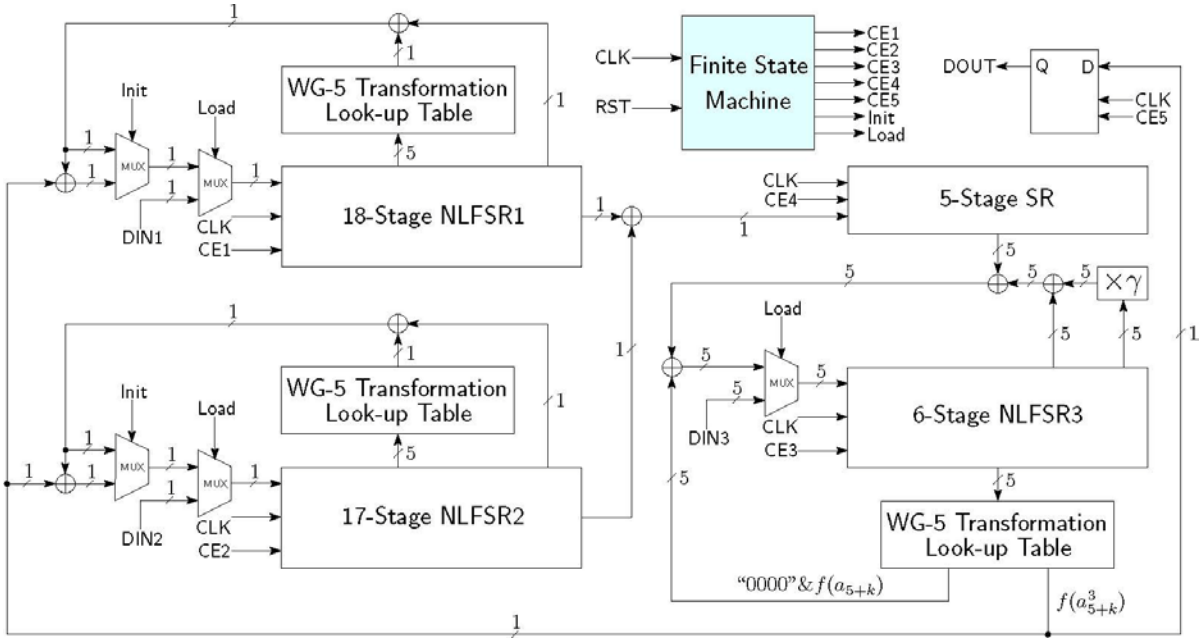

Figure 3. The Data Path of the Lightweight PRNG Core

6.3. Implementation Results and Comparisons

The hardware implementation shows that the PRNG core totally occupies 46 slices (12 and 34 slices for building blocks I and II, respectively) on the target FPGA device and achieves a throughout of 45 Mbps. Table 4 presents a comparison with other PRNGs in terms of hardware implementation and achieved randomness properties. One can notice that our PRNG has a lower hardware complexity than that in [22]. When compared to the PRNG proposed in [20] our design costs a similar number of logic gates with the usage of two NLFSRs replacing the TRNG in [20]. However, if we only compare the hardware implementation cost for the pseudorandom number generator module (i.e., the building block II in our design) in both proposals, our design only needs a half number of logic gates as that in [20]. Although the hardware complexity of our PRNG is slightly larger than that of SPONGENT-80 [5], our design can provide desired randomness properties such as period and linear complexity that cannot be guaranteed by SPONGENT-80.

Table 4. A Comparison with Other PRNGs

Functions	State Size	Area/GE	Devices	Randomness	
				Period	LS
Our PRNG	**65**	46 Slices/ 760 GE (est.)	XC3S50-PQ208	$2^{32.67}$	$2^{18.58}$
LAMED [22]	64	1585 GE	-	-	-
Melia-Segui et al. [20]	16	761 GE	-	-	-
U-QUARK [1]	136	1379 GE	0.18µm CMOS	-	-
DM-PRESENT - 80 [6]	144	1600 GE	0.18µm CMOS	-	-
PHOTON-80/20/16 [14]	100	865 GE	0.18µm CMOS	-	-
SPONGENT-80 [5]	88	738 GE	0.13µm CMOS	-	-

In terms of the time delay for generating the first 16-bit pseudorandom number, our design totally requires 134 clock cycles, including 18 clock cycles for loading key and IV, 36 clock cycles for the initialization, and 80 clock cycles for generating the first 16-bit random number. After that, each 16-bit random number can be obtained every 80 clock cycles. Assuming that the EPC tags run at the clock frequency of 100 KHz and two 16-bit random numbers are needed for the tag identification protocol according to the EPC C1 Gen2 standard, one can identify about 510 tags in one second by using the proposed lightweight PRNG.

Remark 2. In the proposed PRNG, we can update the 45-bit key at the end of each session by generating 45 extra bits in 225 clock cycles and these 45 bits will be loaded at proper aforementioned key positions. This key updating procedure can be used to provide better security. In this way it is possible to generate at least $2^{16.26} \times 2^{20}$ consecutive random numbers for one key and for different IVs.

7. Conclusions

In this paper, we propose a lightweight pseudorandom number generator that is in compliance to the EPC Class-1 Generation-2 standard and has guaranteed randomness properties like period and linear span. Considering the high power-consumption, large area and low throughput of TRNGs, we replace the TRNG used in previous works by a PRNG with good statistical properties. In our design, the pseudorandom sequence is generated using a nonlinear feedback shift register. Moreover, the statistical tests specified by the EPC C1 Gen2 and the NIST standards, algebraic attacks, cube attacks and time-memory-data tradeoff attacks are employed to characterize the security properties of the proposed PRNG and a comparison with the sponge-based PRNGs is conducted. In addition, an FPGA implementation shows that the proposed PRNG can be implemented using 46 slices and can generate a 16-bit random number every 80 clock cycles after an initialization process of 36 clock cycles.

8. References

[1] J. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, "QUARK: A Lightweight Hash" *Cryptographic Hardware and Embedded Systems - CHES 2010*, Vol. 6225, LNCS, Springer-Verlag, 2010, pp. 1 - 15. http://131002.net/quark/.

[2] G.K. Balachandran, and R.E. Barnett, "A 440-nA True Random Number Generator for Passive RFID Tags", *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55 (11), December 2008, pp. 3723 -3732.

[3] G. Bertoni, J. Daemen, M. Peeters, and G.V. Assche. Sponge based Pseudo-random Number Generators, *Cryptographic Hardware and Embedded Systems - CHES 2010*, Vol. 6225, LNCS, pp. 33 – 47, Springer-Verlag, 2010.

[4] A. Biryukov and A. Shamir, "Cryptanalytic Time /Memory/Data Tradeoffs for Stream Ciphers", *Advances in Cryptology-Asiacrypt'00*, Vol. 1976, LNCS, Springer-Verlag, 2000, pp. 1-13.

[5] A. Bogdanov, M. Knevzevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede, "SPONGENT: A Lightweight Hash Function", *Cryptographic Hardware and Embedded Systems - CHES 2011*, Vol. 6917, Springer-Verlag, 2011, pp. 312-325.

[6] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M. J.B. Robshaw, and Y. Seurin, "Hash Functions and RFID Tags: Mind the Gap", *Cryptographic Hardware and Embedded Systems - CHES 2008*, Vol. 5154, LNCS, Springer-Verlag, 2008, pp. 283 - 299.

[7] W. Che, H. Deng, X. Tan, and J. Wang, "A Random Number Generator for Application in RFID Tags", In *Networked RFID Systems and Lightweight cryptography*, Chapter 16, Springer-Verlag, 2008, pp. 279-287.

[8] N. Courtois and W. Meier, "Algebraic Attacks on Stream Ciphers with Linear feedback Shift Registers", *Advances in Cryptology-Eurocrypt'03*, Vol. 2656, LNCS, Springer-Verlag, 2003, pp. 345-359.

[9] I. Dinur, and A. Shamir, "Cube Attacks on Tweakable Black Box Polynomials", *Advances in Cryptology-EUROCRYPT'09*, LNCS, Springer-Verlag, 2009, pp. 278-299.

[10] EPCglobal. "EPC Radio-Frequency Identification Protocol Class-1 Generation-2 UHF RFID for Communication at 860-960 MHz", 2008, http://www.epcglobalinc.org/.

[11] S.W. Golomb and G. Gong, "*Signal Design for Good Correlation: For Wireless Communication, Cryptography, and Radar*", Cambridge University Press, New York, NY, USA, 2004.

[12] G. Gong and A. Youssef, "Cryptographic Properties of the Welch-Gong Transformation Sequence Generators", *IEEE Transactions on Information Theory*, Vol. 48, No. 11, November 2002, pp. 2837-2846.

[13] G. Gong, S. Ronjom, T. Helleseth, and H. Hu, "Fast Discrete Fourier Spectra Attacks on Stream Ciphers", IEEE Transactions on Information Theory, Vol 57, No. 8, August 2011, pp. 5555-5565.

[14] J. Guo, T. Peyrin, and A. Poschmann, "The PHOTON Family of Lightweight Hash Functions", *Advances in Cryptology-CRYPTO'11*, Springer-Verlag, 2011, pp. 222 - 239.

[15] D.E. Holcomb, W.P. Burleson, and K. Fu, "Initial SRAM State as a Fingerprint and Source of True Random Numbers for RFID Tags", In *Proceedings of the Conference on RFID Security (RFIDSec'07)*, July 2007.

[16] A. Juels, "RFID Security and Privacy: A Research Survey", *IEEE Journal on Selected Areas in Communications (J-SAC)*, Vol. 24, No. 2, February 2006, pp. 381-394.

[17] A. Klapper, "Linear Complexity of Finite Field Sequences over Different Fields", *International Workshop on Sequence Design and Applications (IWSDA)*, Fukuoka, Japan, October 2005.

[18] D.E. Knuth, *The Art of Computer Programming*, Volume 2, Seminumerical Algorithms, Addison-Wesley, 1969.

[19] K. Mandal, X. Fan, and G. Gong, "Warbler: A Lightweight Pseudorandom Number Generator for EPC C1 Gen2 Tags", *In Radio Frequency Identification System Security, RFIDSec'11 Asia*, Vol 8, November 2012, pp. 73-84.

[20] J. Melia-Segui, J. Garcia-Alfaro, and J. Herrera-Joancomarti, "Analysis and Improvement of a Pseudorandom Number Generator for EPC Gen2 Tags", *Proceedings of the 14th International conference on Financial Cryptography and Data Security*, FC'10, Springer-Verlag, 2010, pp. 34-46.

[21] W. Meier, and O. Staffelbach, "Fast Correlation Attacks on Certain Stream Ciphers", *Journal of Cryptology*, 1989, pp. 159-176.

[22] P. Peris-Lopez, J.C. Hernandez-Castro, J.M. Estevez-Tapiador, and A. Ribagorda, "LAMED - A PRNG for EPC Class-1 Generation-2 RFID Specification", *Computer Standard Interfaces*, Vol. 31, January 2009, pp. 88-97.

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

[24] V.B. Suresh and W.P. Burleson. "Entropy extraction in metastability-based TRNG", *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST 2010)*, June 2010, pp. 135-140.

[25] H. Wu, and B. Preneel, "Chosen IV Attack on Stream Cipher WG", ECRYPT Stream Cipher Project Report 2005/045, Available at http://cr.yp.to/streamciphers/wg/045.pdf.

[26] Xilinx Inc., "Spartan-3 FPGA Family Data Sheet", DS099, December 4, 2009, available at http://www.xilinx.com/support/documentation/datasheets/ds099.pdf.

