A Practical Implementation Attack on Weak Pseudorandom Number Generator Designs for EPC Gen2 Tags — Source link

Joan Melià-Seguí, Joaquin Garcia-Alfaro, Jordi Herrera-Joancomartí

Institutions: Open University of Catalonia, École nationale supérieure des télécommunications de Bretagne, Autonomous University of Barcelona

Related papers:

- J3Gen: a PRNG for low-cost passive RFID.

- A Random Number Generator for Application in RFID Tags

- LAMED - A PRNG for EPC Class-1 Generation-2 RFID specification

- A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications

- Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers

A Practical Implementation Attack on Weak Pseudorandom Number Generator Designs for EPC Gen2 Tags

Joan Melià-Seguí · **Joaquin Garcia-Alfaro** · **Jordi Herrera-Joancomartí**

Abstract The Electronic Product Code Generation 2 (EPC Gen2) is an international standard that proposes the use of Radio Frequency Identification (RFID) in the supply chain. It is designed to balance cost and functionality. As a consequence, security on board of EPC Gen2 tags is often minimal. It is, indeed, mainly based on the use of on board pseudorandomness, used to obscure the communication between readers and tags; and to acknowledge the proper execution of password-protected operations. In this paper, we present a practical implementation attack on a weak pseudorandom number generator (PRNG) designed specifically for EPC Gen2 tags. We show that it is feasible to eavesdrop a small amount of pseudorandom values by using standard EPC commands and using them to determine the PRNG configuration that allows to predict the complete output sequence.

Joan Melià-Seguí
Universitat Oberta de Catalunya,
Rambla Poble Nou 156, 08018 Barcelona - Spain,
E-mail: melia@uoc.edu

Joaquin Garcia-Alfaro
Institut Telecom, Telecom Bretagne
02, rue de la Chatagneraie, Cesson-Sevigne 35576 - France
E-mail: joaquin.garcia-alfaro@acm.org

Jordi Herrera-Joancomartí
Universitat Autònoma de Barcelona,
Edifici Q, Campus de Bellaterra, 08193, Bellaterra - Spain,
E-mail: jherrera@deic.uab.es

1 Introduction

The Electronic Product Code Generation 2 (EPC Gen2) is an international standard that proposes the use of Radio Frequency Identification (RFID) in the supply chain. It is designed to balance cost and functionality. The development of Gen2 tags faces, in fact, several challenging constraints such as cost, compatibility regulations, power consumption, and performance requirements. As a consequence, the computational capabilities of Gen2 tags are very simple. In this sense, the Gen2 specification only considers two basic on board security features: pseudorandom number generators (PRNGs) and password-protected operations. The pseudorandomness offered by on board PRNGs is, indeed, used to protect the password-protected operations. PRNGs are also used as an anti-collision mechanism for inventorying processes [1]; and to acknowledge other Gen2 specific operations (e.g., memory writing, decommission of tags, and self-destruction). PRNGs are, therefore, the crucial components that guarantee Gen2 security.

The use of weak PRNGs that allow the predictability of the outgoing sequences introduces important security flaws in EPC communications. For example, it allows an attacker to bypass the security of the password-protected commands defined in the Gen2 standard (e.g., the *access* and the *kill* commands [1]). The execution of these commands is processed by the tag if a 32-bit value, stored in the reserved memory of the tag, is provided by the reader. Therefore, if a compliant reader wants to request the execution of these special commands, it must provide an appropriate copy of the 32-bit password values. The security model of the EPC Gen2 specification considers that the reader-to-tag channel requires protection to handle an eventual eavesdropping of these values. For this reason, the reader must obscure the transmission of the 32-bit password values. To do so, the reader requests beforehand two 16-bit pseudorandom sequences to the tag. These two sequences are computed by the on-board PRNG of the tag, and sent as cleartext to the reader (notice that the EPC Gen2 specification does not require protection for the tag-to-reader channel). Once obtained the two pseudorandom sequences, the reader transmits the 32-bit password values in two steps: (1) it obscures the 16 lowest bits of the password by an exclusive OR (XOR) logical operation with the first pseudorandom sequence, and transmits the resulting first half-password; (2) it obscures the remainder 16 bits of the password by XORing them with the second pseudorandom sequence, and transmits the resulting second half-password. Since the attacker is not supposed to be capable of eavesdropping the tag-to-reader channel, the only (practical) way to retrieve the password values must come from the eavesdropping of the reader-to-tag channel. This is indeed possible if the on-board tag's PRNG is predictable. The attacker can apply, for instance, an active scanning of the tags, to analyze the predictability of their pseudorandom sequences; or a passive eavesdropping of reader-to-tag acknowledgments, with the same purpose of analyzing the predictability of tags' pseudorandom sequences. If either attack succeeds, it then suffices to apply a simple XOR operation with the predicted sequences and the contents of the messages transmitted over the reader-to-tag channel to decrypt the remainder ciphertext (e.g., the protected half-passwords).

In this paper, we analyze the proposal presented in [2], in which the authors describe the construction of a cost-effective PRNG for EPC Gen2 devices. We demon-

strate, by implementing a practical attack, that their approach is not secure. An attacker may obtain the PRNG configuration with very few observations and then he is able to derive the whole pseudorandom sequence. Although the attack implementation has been applied to a specific PRNG proposal, the procedure used to obtain the data is based on standard EPC commands and it can be applied to any EPC tag communication to eavesdrop the output of the PRNG.

The paper is organized as follows. Section 2 discusses the challenges of designing PRNGs for EPC Gen2 tags, reviews some proposals and finally describes the suitability of using linear feedback shift registers (LFSRs) for the generation of pseudorandom sequences. The section also describes the proposal presented by Che *et al.* in [2]. In Section 3, we present an analysis of the Che *et al.* scheme. We give the details of a statistical analysis performed over the output data based on the National Institute of Standards and Technology (NIST) statistical test for pseudorandomness. Based on the weakness detected by the NIST test, we also detail an attack that, given a small number of output bits, can determine the whole sequence. Section 4 provides the details of the attack implementation. The section provides information about the tools used to implement the attack and the empirical results obtained in the attack of the Che *et al.* scheme. Finally, Section 5 concludes the paper and gives some ideas for further research.

2 Pseudorandom number generators for EPC Gen2 tag

The design of PRNGs for EPC Gen2 tags is not an easy task due to the computational and memory restrictions that these tiny devices imply. Capabilities of this type of tags are so small that security features for the EPC Gen2 standard are expected to be implemented with a small amount of equivalent logic gates (GE), defined in the literature between 2,000 and 5,000 [3]. This is a extremely small value if we consider that a standard hash function (the most simple cryptographic transformation), like SHA1, needs at least 8,120 GE to be implemented [4].

2.1 Existing proposals

Existing commercial Gen2 tags do implement a PRNG, as it is an EPC standard mandatory, but companies are often reluctant to present the design of their PRNGs. Manufacturers simply refer to testbeds that show the accomplishment of some expected requirements, most of them for compatibility purposes. They fail to offer convincing information about the PRNGs designs [5]. This is mostly security through obscurity, which is always ineffective in security engineering, as it has been shown with the disclosure of the PRNG used in the MIFARE Classic chip [6] that has shown a vulnerable PRNG.

Few PRNG proposals have been presented in the scientific literature specifically designed for EPC tags. To the best of our knowledge, only three papers explicitly propose PRNG for EPC tags. On one hand, Peris-Lopez *et al.* present in [7] a deterministic algorithm that relies on the use of 32-bit keys and pre-established initial

states. The set of functions mainly consists of bit rotations, bitwise operations, and modular algebra, building a 32-bit PRNG. The authors also propose an alternative 16-bit version of their PRNG for EPC Gen2 compatibility. To reduce the output length from 32 to 16 bits, Peris et al. divide the 32-bit output in two halves and XOR them to obtain the 16-bit output sequence. No evidences of further achievements other than hardware complexity and statistical behavior are provided. Moreover, the inherent peculiarity of their construction methodology obscures potential comparison with other designs in the literature. On the other hand, Che *et al.* describe in [2] a hybrid approach that combines the use of Linear Feedback Shift Registers (LFSR) and physical properties to build random sequences (see Section 2.3 for a detailed description of their scheme). A similar idea has been also used in [8] to design a PRNG. In this case the authors handle the inherent linearity of LFSRs by means of a multiple-polynomial approach. The authors present a secure PRNG design suitable to the current EPC Gen2 technology, providing evidences of statistical and hardware compatibility.

2.2 LFSR-based pseudorandom number generators

Linear feedback shift registers (LFSRs) are an important tool for designing PRNG for EPC Gen2 tags. They lead to extremely efficient and simple hardware implementations. For instance, a 16-cell LFSR can be implemented with only 192 GE. A LFSR is a digital circuit that contains a shift register and a feedback function. The shift register is composed of n binary cells that share the same clock signal. Each time a bit is needed, the content of the register is shifted one cell, obtaining the most significant bit of the register in the previous state. The feedback function computes a new bit using some bits of the register, obtaining the less significant bit to be filled in the new state of the register. The feedback function of a LFSR is basically an exclusive OR logical operation (XOR) of some cells content, named *taps*.

Although LFSRs can be implemented efficiently, their main drawback is that their sequences are high predictable [9,10]. For example, let $s_{k+1}, s_{k+2}, \cdots, s_{k+2n}$ be a sequence of $2n$ consecutive bits generated from an LFSR. Let $\mathcal{B} = (b_n, b_{n-1}, \cdots, b_1)$ be the feedback function of the LFSR. Then, the feedback function can be easily computed by solving the following equation system:

$$
\begin{bmatrix}
s_{k+1} & s_{k+2} & \cdots & s_{k+n} \\
s_{k+2} & s_{k+3} & \cdots & s_{k+n+1} \\
\vdots & \vdots & \ddots & \vdots \\
s_{k+n} & s_{k+n+1} & \cdots & s_{k+2n-1}
\end{bmatrix}
\begin{bmatrix}
b_n \\
b_{n-1} \\
\vdots \\
b_1
\end{bmatrix}
=
\begin{bmatrix}
s_{k+n+1} \\
s_{k+n+2} \\
\vdots \\
s_{k+2n}
\end{bmatrix}
\tag{1}
$$

By solving Equation (1) we obtain the feedback polynomial coefficients. Despite the $2^n - 1$ period length generated by a n LFSR, the full sequence can be determined only with $2n$ consecutive bits due to the linearity of the system.

This linearity must be handled before using LFSRs to build robust PRNGs. Several basic constructions can be used to hide linearity, while maintaining suitable statistical properties and long output periods. One of these techniques are filters. Filters

use a non-linear feedback function as an input to the register. The filter should not be too simple to be weak but neither too complex, otherwise it would become the bottleneck of the generator. However recent attacks to the MIFARE PRNG [6] have demonstrated the vulnerability of this kind of generators when the non-linear function is not taken carefully. Another approach to break the linearity of a LFSR is to use a non-linear combination of multiple LFSRs to generate a unique output. Generally the output of one LFSR is used to select or combine the output of one or more LFSRs, in the same or different clock times. Known examples of this approach are the Geffe, A5 or the Shrinking generator [11]. The output generated from this constructions is statistically weak, being vulnerable to correlation or side-channel attacks [12]. Also the irregular output data rate from some of these constructions (e.g. the shrinking generator) is not suitable for PRNG used in security environments. Finally, generators with memory are another alternative. Additional memory can be used to add some non-linear information in between the clock steps of the LFSR. Besides the memory, also binary adders and carry registers should be used to complete this approach.

The different techniques of deterministic modifications of LFSRs explained so far are useful for keystream generators where *sender* and *receiver* can share a secret *k* as a key for the PRNG one-time pad communications. However, the specific communication model of EPC Gen2 systems uses another paradigm where *sender* and *receiver* cannot share any secret *k*. Instead of this, the low-power tag-to-reader communication is used to transmit in plain text the nonces to be used as a keystream for the reader-to-tag communication. This scenario allows other strategies for the linearity avoidance of LFSRs.

A first straightforward strategy is to suppress the LFSR itself and use a true random data source as a random number generator. Although this approach is theoretically sound, implementations of true random number generators obtain their randomness from the device energy and such energy is very scarce in an EPC Gen2 tag. As a result, the generator throughput cannot reach the minimal requirements of the EPC communication standard. Having this problem in mind, Che *et al.* propose in [2] the combination of true random numbers (*trn*) extracted from physical effects on tag, and LFSRs to increase the throughput of the generator while decreasing the predictability of the output sequence. However, as we review later in this paper, their approach does not achieve the objective to break the linearity of the output sequence. In [8], authors also combine true random data and LFSRs to create a PRNG.

A part of these two proposals, there are not many references in the literature that combine true random data and LFSR to obtain a good PRNG. The main reason, as we already stated, is that the obtained PRNG cannot be used as an additive stream cipher for a standard sender-receiver communication model due to the infeasibility of reproducing the same sequence at both communication parts since the cipher sequence will be affected by a true random source.

2.3 The Che *et al.* proposal

In [2], Che *et al.* present a new PRNG for application in RFID tags. Their system relies on an oscillator-based Truly RNG (TRNG), and exploits the thermal noise of
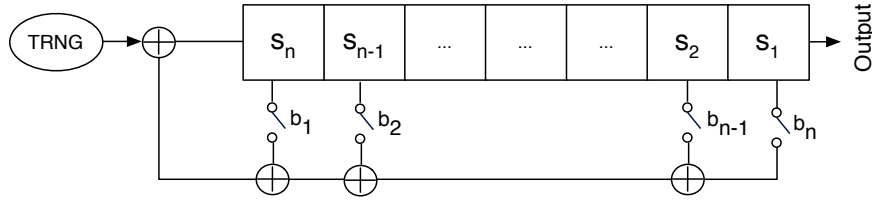
Fig. 1 PRNG scheme based on the Che *et al.* specifications

two resistors to modulate the edge of a sampling clock and generate the true random bits (*trn*). Authors state the final system prevents potential attackers to perform any effective prediction about the generated sequence (even if the design is known) thanks to the white noise based cryptographic key generation.

After describing its TRNG oscillator-based core, the authors focus on design considerations specially regarding *power consumption* and *output data rates* trade-offs. Knowing the fact that the higher the frequency oscillation of the system, the higher the current (thus also power) consumption, the authors look for system level optimization in order to reduce the power consumption due to the low-power restrictions of RFIDs.

The optimization proposed by Che *et al.* relies on the combination of the TRNG and a LFSR (cf. Figure 1). Adding a LFSR to the TRNG lets the system reduce the clock frequency proportionally to the number of cells of the LFSR. Specifically, exploiting the initial state of a 16-bit LFSR combined with the addition of the generated truly random number (*trn*) for each cycle ring, allows the system to decrease the clock frequency with a $\frac{1}{16}$ factor.

Authors claim that [2]: *"If we add 1-bit truly random number in the cycle ring as a random number seed, the output sequence of the LFSR will also be unpredictable and irreproducible as a TRNG."*. We show in the next section that this claim does not hold.

3 Analyzing and exploiting the Che *et al.* proposal

In this section, we present an analysis of the PRNG proposal presented by Che *et al.* described above. We give the details of a statistical analysis, performed over the output data, based on the NIST statistical test for pseudorandomness. Based on these results, we detail an attack that, given a small number of output bits, can determine the whole sequence.

3.1 Che *et al.* statistical analysis

Since the main property of a PRNG is to ensure the forward unpredictability of its generated sequence, the correctness of a PRNG can be measured with statistical tests applied to the output sequence.

Table 1 Che *et al*. results for the NIST statistical test suite

Sequence	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
Frequency	0.99	0.98	0.96	1.00	1.00	0.97	0.97	0.99	0.96	0.97
BlockFrequency	1.00	1.00	0.97	0.98	1.00	0.98	1.00	0.99	0.98	0.99
Runs	0.98	1.00	1.00	0.99	0.99	0.99	0.96	0.98	0.98	1.00
LongestRun	**0.96**	**0.96**	0.98	0.97	**0.94**	0.96	**0.95**	0.98	0.99	**0.94**
Binary Matrix Rank	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
OverlappingTemplate	0.96	**0.94**	0.99	0.98	0.98	**0.93**	0.97	0.98	0.98	**0.95**
Universal	1.00	0.99	0.98	0.99	1.00	0.97	0.99	0.99	0.99	0.98
ApproximateEntropy	0.99	0.97	1.00	0.99	0.98	0.98	0.99	1.00	1.00	1.00
LinearComplexity	0.99	1.00	0.99	0.99	**0.95**	1.00	1.00	0.99	0.99	1.00
CumulativeSums	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$
NonPeriodicTemplate	$\frac{148}{148}$	$\frac{148}{148}$	$\frac{148}{148}$	$\frac{148}{148}$	$\frac{148}{148}$	$\frac{148}{148}$	$\frac{148}{148}$	$\frac{148}{148}$	$\frac{147}{148}\star$	$\frac{148}{148}$
RandomExcursions	$\frac{7}{8}\star$	$\frac{7}{8}\star$	$\frac{8}{8}$	$\frac{7}{8}\star$	$\frac{8}{8}$	$\frac{7}{8}\star$	$\frac{8}{8}$	$\frac{6}{8}\star$	$\frac{8}{8}$	$\frac{8}{8}$
RandomExcursionsVariant	$\frac{18}{18}$	$\frac{18}{18}$	$\frac{18}{18}$	$\frac{18}{18}$	$\frac{18}{18}$	$\frac{18}{18}$	$\frac{18}{18}$	$\frac{18}{18}$	$\frac{18}{18}$	$\frac{18}{18}$
Serial	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$	$\frac{2}{2}$

We take the National Institute of Standards and Technology (NIST) suit test for checking the randomness deviations of a binary random sequence [13]. NIST testing algorithms use a hypothesis test considering the randomness of the sequence as the *null hypothesis* H_0, and the non-randomness as the alternative hypothesis, H_a. Tests are performed regarding a level of significance or critical value, denoted as α hereinafter.

NIST tests produce *P-values* summarizing the strength of the hypothesis. If *P-values* $\geq \alpha$, H_0 is accepted. It is not necessary that strictly all *P-values* hold this bound for the sequence to be considered as a good pseudorandom sequence. In fact, the NIST recommends that the proportion of test over the significance level, must fit in the interval

$$\hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}} \tag{2}$$

where $\hat{p} = 1 - \alpha$, and *m* is the sample size. A common value used in cryptography [13] to statistically confirm the randomness of the analyzed data would be $\alpha = 0.01$, that means one would expect 1 in 100 sequences to be rejected. *P-values* passing α give a confidence of 99.9% of the randomness of the evaluated sequence (if 100 sequences are evaluated, results should pass 0.9615 as defined in Equation 2).

In order to evaluate the randomness quality of the sequence produced by the Che *et al.* scheme, we have generated 230 MB of output data from an implementation of their proposed PRNG. Such data has been divided in ten different data sequences (T_i) that have been independently analyzed using the NIST suit tests.

NIST test results for the Che *et al.* random generated data are presented in Table 1. Each column represents 23 MB of pseudorandom data generated with different *seed* and true random source. Each row refers to a test included in NIST test suite. The first nine tests are represented with the numerical value of the uniformity of *P-values*. The last five tests are in fact a set of different tests thus in order to represent each of the

values, an achievement ratio is represented following the same decision rule of the first tests (Equation 2). Tests refusing randomness hypothesis are denoted with bold letters in the table. For tests consisting on a set of tests, an asterisk is added when some of the tests are not successfully achieved.

Results show a statistical evidence of non randomness for the *Binary Matrix Rank Test* (cf. Table 1). Such test constructs binary matrices from the analyzed data and checks for linear dependence among the rows or columns of the constructed matrices. The fact that the *Binary Matrix Rank Test* fails for all the sequences, gives a clear evidence of a non-randomness due to linearity problems.

3.2 Exploiting the linearity weaknesses of the scheme

As we have pointed out in Section 2.2, the main vulnerability of a PRNG based on a linear feedback shift register comes from its easy predictability due to its linearity properties.

Results presented in Section 3.1 show that the Binary Matrix Rank Test from the NIST statistical test suite fails for the Che *et al* scheme, providing information that the scheme does not succeed in breaking the linearity of the underlying LFSR. In fact, a specific attack to break the Che *et al.* PRNG based on the inherent linearity of the LFSR has been presented in [8] and is next briefly described.

Notice that in the Che *et al.* scheme the pseudorandom sequence is produced by a LFSR XORed in its first cell with a *truly* random bit (cf. Figure 1). That means we can find a $2n$ pseudorandom output sequence of the proposed scheme identically equal to the one of the $n-$bit LFSR (without of the XORed true bit) in case that 2 consecutive random bits are 0. Such event will occur with probability $1/4$ assuming bits are true random.

3.2.1 Attack description

Our scenario is composed by a Che *et al.* system that produces pseudorandom bits. Only a part of the pseudorandom output sequence, denoted by s_a is known to the attacker, besides the size n of the LFSR. On the other hand, the *seed* (initial state) and the feedback polynomial coefficients remain secret to the attacker. The attack will succeed if the attacker can provide the LFSR feedback polynomial (cf. Figure 2).

To generalize the attack, we also assume that the attacker cannot determine the first bit of the sequence, that means he has no information if a given s_a sequence, with $|s_a| = 2n$ (the length of the sequence), has been affected by exactly two trn values (that means the attacker finds two exact LFSR rounds) or the sequence has been modified by three trn values.

With probability $\frac{1}{n}$, the sequence, s_a with $|s_a| = 2n$ has been affected by exactly two trn and, in this case, the probability to obtain the $2n$ values of the LFSR despite the XORed trn is $\frac{1}{4}$ (two consecutive zeros). That means that, with probability $\frac{1}{4n}$, we can obtain $2n$ values of the LFSR that composes the system and with this sequence we are able to compute the feedback polynomial and the whole pseudorandom sequence.
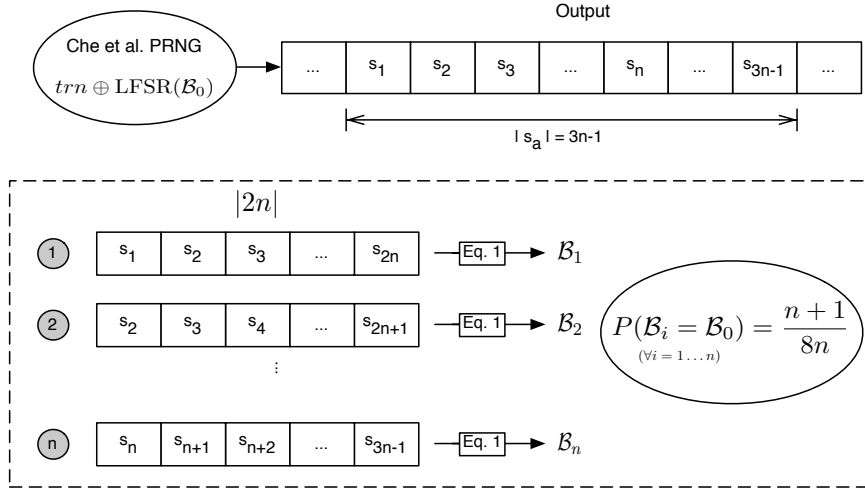
Fig. 2 Attack scheme to the Che *et al.* PRNG

Now, assume that $|s_a| = 3n - 1$. If the sequence is divided into n subsequences of length $2n$, we can ensure that one of these subsequences has been affected by exactly two trn. The remainder $n - 1$ subsequences, have been affected by three trn. However, notice that if the three trn are zeros, the n vectors of length $2n$ will give the same feedback polynomial. The probability of such event is $\frac{1}{8}$. Then, from this fact, we can derive Equation 3 which provides the probability of success of an attack that analyzes a sequence with $|s_a| = 3n - 1$:

$$P_{\text{success}}(3n - 1) = \frac{1}{4}\left(\frac{1}{n}\right) + \frac{1}{8}\left(\frac{n-1}{n}\right) = \frac{n+1}{8n} \tag{3}$$

Obviously, the probability of success increases with $|s_a|$ since increasing the $|s_a|$ implies that more trn bits affect the sequence and then the probability of finding three consecutive zeros also increases. Figure 3 shows the probability of success of an attack with s_a length for a particular system with a LFSR of length $n = 16$. Notice that only 160 bits ($10n$) are enough to perform a successful attack with probability higher than 50%, and 464 bits ($29n$) implies more than a 90% of success probability.

3.2.2 Obtained results

To test the correctness of the theoretical evaluation, the described attack has been implemented over the same ten pseudorandom sequences (T_i) used to execute the NIST tests (cf. Section 3.1).

The first analysis validates that the probability of finding the feedback polynomial matches the one described in Equation 3. In this case, the algorithm takes $|s_a| = 3n - 1$ bits from T_i starting at a random position and tries to attack the system by finding n equal feedback polynomials. The operation is repeated one thousand times
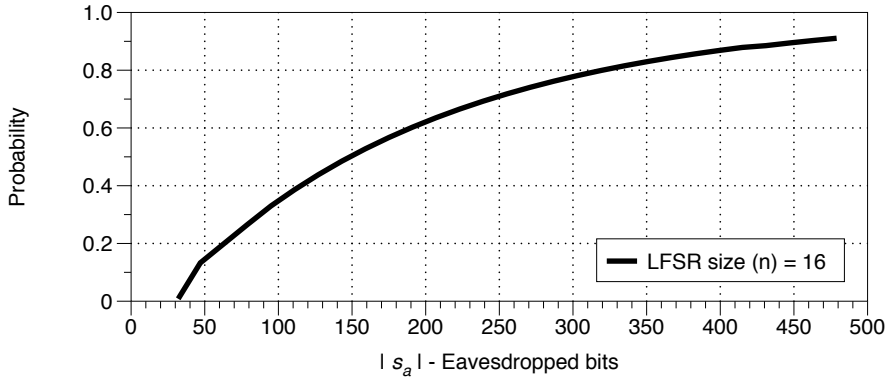
Fig. 3 Reliability on the Che *et al.* attack regarding $|s_a|$

Table 2 Attack success rate for $|s_a| = 3n - 1$

Sequence	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
attack success (%)	0.132	0.137	0.131	0.126	0.139	0.137	0.129	0.137	0.138	0.128

Table 3 Value of $|s_a|$ for a successful attack in the worst case after 10 tests

Sequence	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
$|s_a|$	238	254	254	190	510	158	254	286	238	222

for each test sequence T_i. Attack success rates are reported in Table 2. Notice that they are close to the theoretic value $\frac{(n+1)}{8n}$ with $n = 16 \approx 0,132$.

The second analysis provides the number of bits needed to achieve a successful attack. Ten different attacks have been performed for every T_i data sequence taking the first bit of s_a at random. Results presented in Table 3 show the number of bits for a successful attack in the worst case, that is the attack that needs a major number of bits. Notice that, although taking the worst case, the number of bits is significantly lower than the whole period $2^{16} - 1$.

4 Attack implementation and empirical results

In this section, we present the RFID devices used to implement the attack. We also describe the implementation scenario and the techniques used to eavesdrop the PRNG from the RFID communication. Finally the practical results are presented.

4.1 Background on the IAIK UHF demo tag

The *IAIK UHF demo tag* [14] is a programmable device intended for developing new commands or functionalities to the EPC Gen2 standard. It allows, moreover, to verify

the new functionality using compliant EPC Gen2 readers. We use this prototype to demonstrate and validate the concepts discussed in the previous sections.

The demo tag consists of four main components: an antenna, a radiofrequency (RF) front-end, a programmable microcontroller, and a firmware library. The antenna captures the energy emitted by the reader and powers up the RF front-end of the tag. The RF front-end demodulates the information encoded in the signal. The resulting data feeds the programmable microcontroller which, in turn, computes a response. To compute the response, the programmable microcontroller executes a software implementation of the EPC Gen2 protocol, implemented as a firmware library. The response is then modulated by the RF front-end and backscattered to the reader. We present in the sequel a condensed background on these four components. More details can be obtained in [14, 15].

Antenna and RF front-end

The antenna connected to the RF front-end consists of a 17cm dipole antenna. The RF front-end utilizes a two-stage charge-pump rectifier to perform amplitude-shift keying (ASK) demodulation. It demodulates the information stored in the signal transmitted on the reader-to-tag channel. It does, indeed, rectification, voltage multiplication, and envelope detection all at once [15]. The power extracted by the rectifier from the RF field emitted by the reader from most compliant EPC Gen2 readers amounts to about 2.4mW. Since this is not enough to power the microcontroller, the demo tag adopts a semi-passive approach, meaning that although the analog parts are powered by the energy harvested from the reader, the digital parts (e.g., the programmable microcontroller) are powered by an external power supply or by an on-board battery. The backscattering of the information computed by the programmable microcontroller consists of the reflected power of the antenna. This power is indeed generated according to the transmitted data. The RF front-end of the demo tag combines both ASK and PSK (phase-shift keying) to modulate information. The backscattering components used by the demo tag to modulate the tag-to-reader signals consist of a resistor, a capacitor, and a fast-switching transistor placed close to the antenna. These components are controlled by the programmable microcontroller.

Programmable microcontroller and firmware Library

The programmable microcontroller connected to the RF front-end of the demo tag consists of an Atmel AVR ATmega128 [16]. It contains all the logic and memory necessary for the demo tag. The ATmega128 is an 8-bit microcontroller based on the AVR architecture. The memory banks of the microcontroller, 128KB of flash memory and 4KB of data memory, can be addressed by three independent 16-bit registers. In addition, the ATmega128 has 32 registers of 8-bits. All 32 registers can act as the destinations of the ATmega128 arithmetic operations. The microcontroller operates exactly one instruction per clock cycle, at frequencies up to the order of 16MHz. An external crystal oscillator connected to the demo tag provides the 16MHz signal to the microcontroller. Three main signals connect the microncontroller to the RF

front-end. A first signal, called DEMOD, provides the demodulated ultra high frequency (UHF) signal from the reader-to-tag channel. A second signal, called MOD, allows the ATmega128 to control the backscatter used to generate the tag-to-reader responses. Finally, a third signal, called RF ON, provides a boolean value to detect the presence of the RF field.

The original IAIK UHF demo tag already provides an appropriate implementation of the EPC Gen2 protocol for the ATmega128. The protocol is implemented as a firmware library stored in the flash memory of the microcontroller. This library contains all the functions necessary to process the readers' standard queries and to compute the appropriate responses. The microcontroller is connected, via an UART module, to a serial-interface connector. This serial interface allows to interact with the demo tag, to provide basic operations such as memory mapping, EPC Gen2 values' configuration, visualization of queries and responses exchanged with compliant readers, and execution of user defined operations. This latter allows to complement the original protocol implementation with new functionalities defined at a user level. By using the JTAG connector provided by the demo tag, it is possible to upload new functionalities to the flash memory of the microcontroller, as well as to perform program debugging. A combination of C code and assembly code can be used to complement or modify the original firmware library. A JTAG download cable allows the transfer of new functionalities or firmware updates. Some other modules connected to the demo tag allow more complex programming possibilities, such as FPGA-based UHF protocol implementations. We refer the reader to [14, 15], and citations thereof, for more information.

4.2 Che *et al.* implementation and experimental setup

In Section 3.2, we have seen how to attack the pseudorandom number generator proposed by Che *et al.* once a sufficient number of pseudorandom values are collected. We show in this section the results of a practical attack against the vulnerable scheme on a real Gen2 setup. The attack is based on the eavesdropping of the communication between a standard EPC Gen2 reader and the demo tag. Indeed, we show how it is possible to obtain an appropriate set of random queries generated by an on-board PRNG, based on the Che *et al.* scheme, to eventually predict the generation of pseudorandom sequences that will be generated later over the demo tag. Figure 4 shows our experimental setup.

The Che *et al.* scheme has been implemented in ANSI C using the Crossworks IDE for AVR from Rowley Associates [17]. The original scheme provided in [2] has been adapted into a code-optimized EPC Gen2 version that can be executed over the microcontroller of the IAIK UHF demo tag. Arithmetic efficient functions such as *bit shifts*, logic operators (*AND*, *OR* and *XOR*) and *modulo 2*, are used to implement the LFSR in the demo tag [11]. The *trn* addition is extracted from the less significant bits of the analogical to digital conversion in the demo tag's microcontroller. Since the generation of pseudorandom sequences is a mandatory operation specified in the EPC Gen2 protocol, an existing PRNG function is already included in the original firmware. By using the Crossworks IDE, we code and merge the PRNG based on the
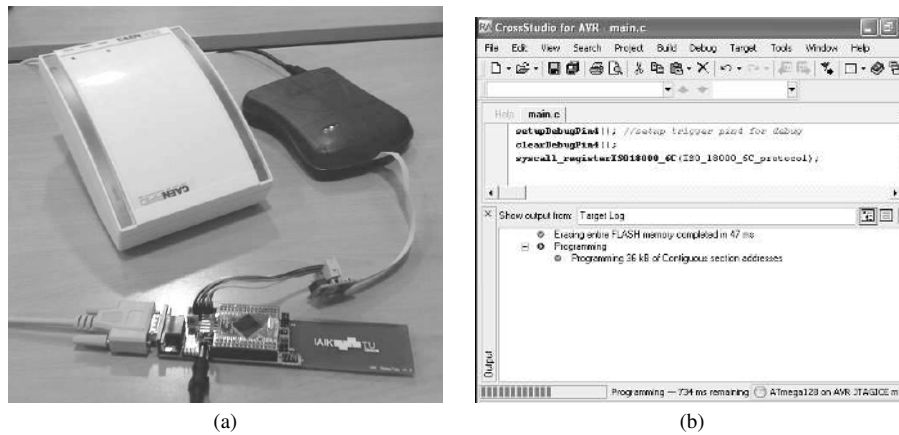
(a)   (b)

Fig. 4 Experimental setup. In (a), we can see the CAEN A829EU Reader, the AVR JTAG MKII Programmer, and the IAIK Graz UHF Demo Tag. In (b), we can see the Crossworks IDE GUI for AVR, uploading the updated firmware over the demo tag

Che *et al.* scheme with the general firmware library to replace the existing PRNG. The JTAG programmer that we use to transfer and to debug the updated firmware merged with the new PRNG implementation is an AVR JTAG MKII programmer [16]. The queries are generated from a standard RFID reader according to EPC Gen2. The RFID reader we use is a short-range reader CAEN A829EU [18]. The reader is controlled by a desk computer over a USB serial port. For the generation of queries, we use a .NET application that controls the communication process with the reader. This application enable us to generate the set of queries required to proceed with the eavesdropping attack. Finally, we use Matlab to decode the set of responses generated over the demo tag. This operation enable us to isolate the pseudorandom queries computed at the demo tag. When the number of sequences collected by the application reaches an appropriate threshold, it proceeds to execute the implementation of the attack we presented in Section 3.2. We provide in the sequel further details about the collection of pseudorandom sequences and the practical results.

4.3 Eavesdropping of pseudorandom sequences and practical results

Due to the Gen2 RF power range characteristics, a realistic attack should only consider reader-to-tag queries because they are much easier to be eavesdropped [5]. Some reader-to-tag queries include pseudorandom sequences (hereinafter denoted

Table 4 Minimum number of RN16s involved in EPC Gen2 operations

Operation	Inventory	Access			
Command	Identification	Read	Write	Lock	Kill
Number of RN16s	1	2	8	2	4

(a) EPC Gen2 *write* protocol.
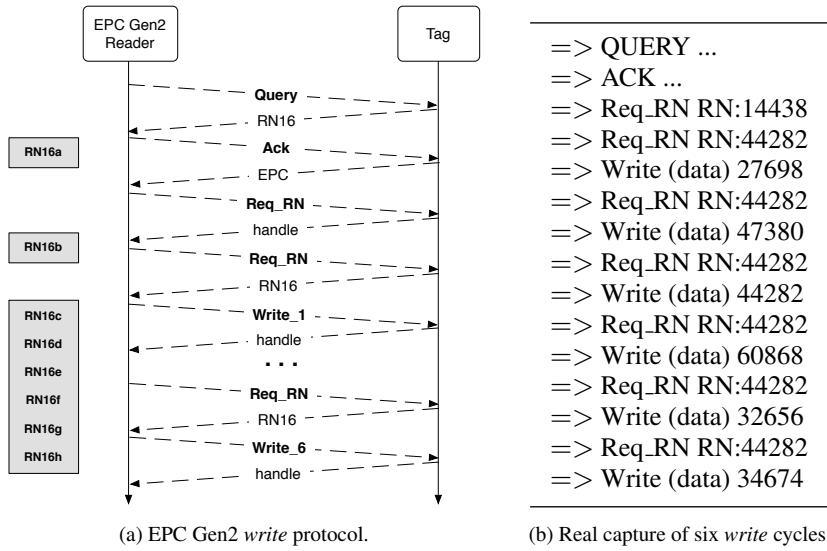
(b) Real capture of six *write* cycles.

Fig. 5 *Write* process for EPC Gen2 and the PRNG utilization. In (a), we can see the six cycles of the EPC Gen2 *write* command. In (b), we can see a real sample of six *write* cycles captured from the reader-to-tag channel

as RN16s) that are computed from the on-board PRNG included on the EPC tags. Table 4 shows the mandatory operations for Gen2 reader-to-tag protocol and the minimum number of RN16s involved in each operation. Notice that the *write* command generates a minimum of eight RN16s for its proper execution. For a full EPC code writing, up to six RN16s must be generated to cover the reader-to-tag communication, besides the two previously generated pseudorandom sequences for the inventory query and the handle descriptor [1].

A *write* operation is an access command used to modify specific areas of a Gen2 tag memory. The reader first identifies the tag with *select* and *inventorying* commands (what shifts the tag from *ready* to *acknowledged* state). Once the tag is *acknowledged* (meaning that the tag has sent its EPC identification) the reader requests a new RN16 to the tag for establishing an *access* session. The new RN16 (denoted as handle) acts as a session key, and must be used to link all the *access* actions to a specific tag. Let us observe that all access commands can be executed both in the *open* or *secured* tag state [1]. If the accessed tag is in the secured state, it means a 32-bit password (exchanged as two 16-bit half-passwords XORed with two RN16s) is necessary to allow the reader to access the tag. In our experiments, we assume that the tag is in the open state, i.e., we do not consider the capture of PRNGs derived from the exchange of the two half-passwords. In this way, an inventoried tag transitions directly to the access mode. For a *write* operation once the reader gets the handle, it initiates a round of writes of 16-bit data sequences (obscured with previously requested RN16s) to the tag. Thus, if a new EPC identification is written to the tag, six $\widehat{write}$ cycles are performed, as we picture in Figure 5(a). The eight generated RN16s represent 128
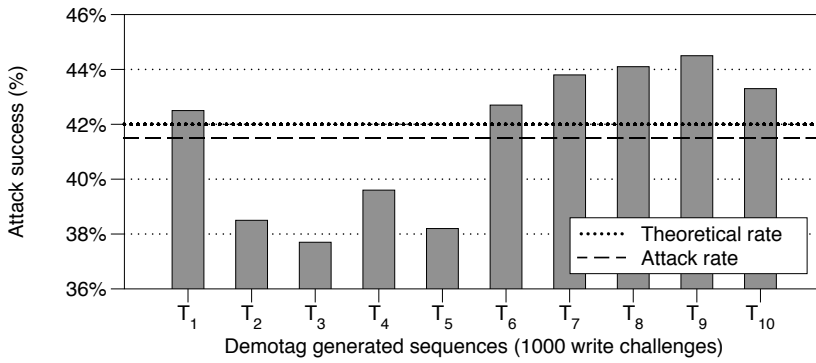
Fig. 6 Che *et al.* PRNG attack success for real Gen2 environment

consecutive bits generated from the PRNG of an EPC Gen2 tag, as specified in the standard [1].

As we pointed out in Section 3.2, the Che *et al.* scheme can be predicted with a reasonable small amount of data. We can now demonstrate this property in our real Gen2 environment, by simply performing an appropriate series of *write* challenges to the adapted Che *et al.* PRNG implemented over the demo tag, and analyzing the resulting RN16s. More precisely, we show that by simply collecting 128 bits (generated from a series of eight RN16s associated to each *write* challenge) is enough to obtain the feedback polynomial of the LFSR with a confidence of about 42%. This value is consistent with the analytical results we anticipated in the previous section, and that are depicted in Figure 3. Figure 5(b) shows a simple example where six write cycles are captured. These captures allow us to collect 96 pseudorandom bits generated from the on-board Che *et al.* PRNG. The sequences are parsed from the matlab code that we feed with the serial interface output of the demo tag. Only reader-to-tag challenges are shown. The reader writes the EPC identification to 0 ($EPC_{96b} = 0_{i\ b...(i+16)\ b} \oplus RN16$), to obtain the RN16s directly from the ciphered data field of the *write* challenges.

The complete set of experiments that we summarize in Figure 6 consists of ten series of *write* commands. Each of these series generates a total of 1,000 *write* challenges from the A829EU reader to the demo tag. As a result, 8,000 RN16s, i.e., 128,000 pseudorandom bits, are captured in total. These pseudorandom bits are computed from the Che *et al.* PRNG implementation deployed over the demo tag. Once stored, the pseudorandom sequences are processed by the matlab code that contains the attack implementation. Let us recall that the attack applies the analysis of the linearity relation for each single *write* challenge. We show that the attack finds the appropriate feedback polynomials of the LFSR each 128 bits with a total ratio of success of 41.5%. This result is very close to the 42% that we predicted in Section 3.2 (Figure 3). Therefore, we are able to confirm the vulnerability of the Che *et al.* PRNG for Gen2 environments.

5 Conclusions

Pseudorandom number generators (PRNGs) are the crucial components that guarantee the confidentiality of EPC Gen2 [1] RFID communications. In this paper, we have described the problems of using linear feedback shift registers (LFSRs) as underlying mechanisms for the implementation of low-cost PRNGs. Without appropriate measures that increase their cost, the linearity of LFSR-based PRNGs lead to insecure implementations. We have analyzed a cost-effective PRNG proposal for EPC Gen2 devices presented by Che *et al.* [2]. The proposal combines thermal noise signal modulation and an underlying LFSR. We have indeed demonstrated that the proposal does not handle properly the inherent linearity of the resulting PRNG. We have described an attack to obtain the feedback polynomial function of the LFSR. This allows us to synchronize and to predict the resulting sequences generated by the Che *et al.* PRNG. We have presented the implementation of a practical attack in a real EPC Gen2 scenario. By means of a compatible Gen2 reader, and a programmable Gen2 tag [14] implementing the Che *et al.* PRNG, we have shown that an attacker can obtain the PRNG configuration with a confidence of 42% by only eavesdropping 128 bits of pseudorandom data. Although the attack implementation has been applied to a specific PRNG proposal, the procedure used to obtain the data is based on standard EPC commands and it can be applied to any EPC tag communication to eavesdrop the output of the PRNG. Future work goes towards designing an efficient and robust PRNG that fulfills the specific restrictions of EPC Gen2 tags, following the work presented in [8].

References

1. EPCglobal. (2008). EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860-960 MHz. `http://www.epcglobalinc.org/standards/`.Accessed 15 July 2010.
2. Che, W., Deng, H., Tan, X., and Wang, J. (2008). Chapter 16, A Random Number Generator for Application in RFID Tags. In Cole, P.H. and Ranasinghe, D.C. (Eds.), *Networked RFID Systems and Lightweight Cryptography* (pp. 279–287). Berlin: Springer-Verlag.
3. Ranasinghe, D.C. and Cole, P.H. (2008). Chapter 8, An Evaluation Framework. In Cole, P.H. and Ranasinghe, D.C. (Eds.), *Networked RFID Systems and Lightweight Cryptography* (pp. 157–167). Berlin: Springer-Verlag.
4. Feldhofer, M. and Rechberger, C. (2006). A Case Against Currently Used Hash Functions in RFID Protocols. In Meersman, R. *et al.* (Eds.), *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops* (pp. 372–381). Berlin: Springer-Verlag.
5. Peris-Lopez, P. (2008). Lightweight Cryptography in Radio Frequency Identification (RFID) Systems. PhD Thesis.`http://www.lightweightcryptography.com/`. Accessed 15 July 2010.
6. Garcia, F., Koning, G., Muijrers, R., van Rossum, P., Verdult, R., Wichers R. and Jacobs, B. (2008). Dismantling MIFARE Classic. In Jajodia, S. and Lopez, J. (Eds.), *Computer Security - ESORICS 2008* (pp. 97–114). Berlin: Springer-Verlag.

7. Peris-Lopez, P., Hernandez-Castro, J., Estevez-Tapiador, J. and Ribagorda, J. (2009). LAMED A PRNG for EPC Class-1 Generation-2 RFID specification. *Computer Standards & Interfaces*, 31(1), 88–97.

8. Melia-Segui, J., Garcia-Alfaro J. and Herrera-Joancomarti, J. (2010). Analysis and Improvement of a Pseudorandom Number Generator for EPC Gen2 Tags. In Curtmola, R. *et al*. (Eds.), *Financial Cryptography and Data Security 2010 Workshops, LNCS* (pp. 34–46). Berlin: Springer-Verlag.

9. Herlestam, T. (1995). On Functions of Linear Shift Register Sequences. *Advances in Cryptology EUROCRYPT 85, LNCS*. doi: 10.1007/3-540-39805-8.

10. Chen, C.L. (1986). Linear Dependencies in Linear Feedback Shift Registers. *IEEE Transactions on Computers*, C-35(12), 1086-1088.

11. Schneier, B. (1996). *Applied Cryptography*. John Wiley & Sons.

12. Joux, A. (2009). *Algorithmic Cryptanalysis*. Chapman & Hall/CRC, Taylor & Francis Group.

13. National Institute of Standards and Technology. (2008). Random number generation. `http://csrc.nist.gov/groups/ST/toolkit/rng/`. Accessed 15 July 2010.

14. SIC, Stiftung Secure Information and Communication Technologies. (2009). UHF RFID Demo Tag. `http://jce.iaik.tugraz.at/sic/products/rfid_components`. Accessed 15 July 2010.

15. M. Aigner *et al.* (2007).   BRIDGE — Building Radio frequency IDentification for the Global Environment. Report on first part of the security WP: Tag security (D4.2.1). `http://www.bridge-project.eu/`. Accessed 15 July 2010.

16. Atmel Corporation. (2009). `http://www.atmel.com/`. Accessed 15 July 2010.

17. Rowley Crossworks IDE. (2009).   Crossworks v1.4 and v2.0 for AVR. `http://www.rowley.co.uk/`. Accessed 15 July 2010.

18. CAEN RFID. (2009). `http://www.caen.it/rfid`. Accessed 15 July 2010.