IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 3, 2017, accepted October 31, 2017, date of publication November 15, 2017,
date of current version December 22, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2774105

A Lightweight Pseudorandom Number Generator for Securing the Internet of Things

AMALIA BEATRIZ ORÚE LÓPEZ, (Member, IEEE), LUIS HERNÁNDEZ ENCINAS, AGUSTÍN MARTÍN MUÑOZ, AND FAUSTO MONTOYA VITINI
Department of Information and Communications Technologies, Institute of Physical and Information Technologies, CSIC, 28006 Madrid, Spain

Corresponding author: Amalia Beatriz Orúe López (amalia.orue@iec.csic.es)

This work was supported in part by the Ministerio de Economía y Competitividad, Spain, under Project TIN2014-55325-C2-1-R (ProCriCiS), in part by the Comunidad de Madrid, Spain, under Project S2013/ICE-3095-CM (CIBERDINE), and in part by the European Union FEDER Funds.

ABSTRACT Lightweight cryptography aims to address the security demands in resource-constrained hardware and software environments, such as the Internet of Things (IoT). These constraints severely limit solutions offered by conventional cryptographic primitives, which turn too expensive to achieve. In this paper, a lightweight pseudorandom number generator that fits the IoT demands is presented. It has a good performance on Atmel 8-bit AVR and Intel Curie 32-bit microcontrollers. The analysis of the hardware complexity in terms of gate equivalent confirms that it is suitable for the IoT.

INDEX TERMS Security, random number generator, internet of things, lightweight cryptography, microcontrollers.

I. INTRODUCTION

The Internet of Things (IoT) paradigm is based on the interconnections among ubiquitous and highly heterogeneous networked "things" such as sensors, actuators, smartphones, etc., whose pervasive use can be attributed to the advanced developments in communications, sensor technologies and networking capabilities, mobile devices, cloud computing, etc. Since in many cases the IoT is related to user's daily life, security and privacy requirements must be fulfilled [1]. However, traditional security measures cannot be directly applied to the IoT due to the different standards and communication technologies involved. The large number of resource-constrained nodes typically employed in the IoT requires the use of lightweight cryptographic primitives.

Lightweight cryptography is composed of a group of cryptographic primitives customized for constrained environments [2]. The term resource-constrained environment is used to describe a development platform that has a reduced design space. For example, in hardware implementations, chip size, energy consumption (e.g. battery life), hardware memory, computation latency, communication bandwidth, etc., should be considered to evaluate the lightweight properties. All hardware lightweight primitive implementations attempt to fulfil their essential functionalities using the minimum size of hardware dice. The efficiency of the implementation depends on the design complexity, the technology used, the throughput, and the energy consumption. The hardware complexity is determined by the number of logic gates required to implement the pseudorandom number generators (PRNGs). It is measured by means of the "gate equivalent" (GE). In CMOS technology, one GE stands for the silicon area occupied by a NAND2 gate; therefore, it is used intuitively to express the chip area of a design. GE has the particularity that different technologies and standard-cell libraries lead to different results.

In the case of software implementations, it is necessary to pay attention to the execution time, the RAM consumption (RAM footprint) and the code size. As a consequence, designers are limited by the available resources and often choose a minimalist approach, without considering the security risks this could represent for the potential users.

True Random and Pseudorandom Number Generators (TRNG and PRNG, respectively) are two of the most important building blocks of cryptosystems. They are used to generate confidential keys, challenges, and nonces. They are also employed in authentication protocols and even in countermeasures against hardware attacks [3]. In the constrained devices of the IoT applications, cryptographically secure PRNGs are difficult to attain due to hardware/software limitations. Only a few descriptions of these PRNGs can be found in the reviewed literature, and some of them have security issues [4].

This work proposes Arrow, a lightweight PRNG that belongs to the Trifork's PRNG family [5], whose structure is suited for resource-constrained devices. The structure of Arrow consists of two coupled Lagged Fibonacci Generators (LFG), mutually scrambled, suitable to secure the great majority of the IoT applications, like those used in smart cards, Radio Frequency Identification (RFID) tags, and wireless sensor nodes.

The rest of this paper is organized as follows. In section II the most important published works about the use of PRNGs in constrained devices are reviewed. Our proposed PRNG, Arrow, is presented in depth in section III. The characteristics of its implementation in different platforms, and some results are discussed in section IV. Conclusions are presented in section V.

II. RELATED WORK

In the reviewed literature, several lightweight cryptographic primitives have been proposed to provide security to resource-constrained devices. We especially focus on cryptographically-secure PRNG designs for resource-constrained devices.

Francillon and Castelluccia [6] designed and implemented TinyRNG, a cryptographic pseudorandom number generator which uses the received bit errors as a source of randomness in wireless sensor nodes. The authors based their implementation on MICA2 motes. Lo Re *et al.* [7] presented an improved version of the TRNG proposed in [8] that uses measurements obtained from the wireless sensor nodes as the sources of physical randomness. Their method uses a distributed leader election algorithm for selecting the random source of data. Lo Re *et al.* implemented the proof of concept using 60 TelosB nodes deployed within an area of 25×15 m^2. Additionally, the robustness of the TRNG algorithm against several attacks was evaluated.

The Warbler PRNG family for low-cost smart devices such as sensor nodes was presented in [9]. It is based on the combination of modified *de Bruijn* blocks and a Welch-Gong (WG) nonlinear feedback shift register. Two instances of Warbler with different security levels, implemented using a 65 nm CMOS process, were presented. The two proposed instances are suitable for securing low-cost smart devices. In [10], an important distinguishing attack against the whole WG family of stream ciphers shows that almost every member of this family is vulnerable to linear attacks; this could represent a threat to the security of Warbler PRNGs.

A PRNG named LAMED was presented in [11] for RFID tags applications. Its design is based on a Genetic Programming algorithm and has an internal state of 64 bits, with a 32-bit key and a 32-bit initial vector. Modular algebra, bitwise XOR operations and bit rotations are the basic operations used for updating the internal state of the PRNG. Two versions of the generator were proposed. The first one is a 32-bit PRNG, and the second one is a 16-bit PRNG for EPC Gen2 compatibility. The NIST, ENT and Diehard statistical test suites were used to validate the randomness of the generators. The hardware complexity analysis of both versions of the generator confirms that it meets the requirements imposed by low-cost technology, i.e., GE $\leq$ 4000 gates, [12], [13].

In [14], the J3Gen PRNG was proposed, based on the previous work in [15]. The J3Gen combines a thermal-noise TRNG and a Dynamic Linear Feedback Shift Register (DLFSR) of n cells, and has four main blocks: an oscillator-based TRNG, a DLFSR architecture, a Decoding Logic, and a Polynomial Selector. The approximate hardware complexity of this PRNG, in terms of GE, is suitable for constrained devices. The security equivalent key size is 372 bits. The J3Gen PRNG was successfully cryptanalyzed by Peinado *et al.* [4] who showed the vulnerabilities of the algorithm by means of a probabilistic attack and a deterministic attack. The former allows to recover the set of feedback polynomials, which constitute the secret information of the PRNG. The latter allows the attacker to reconstruct the entire output sequence of the PRNG by the knowledge of only a few bits of the sequence.

III. ARROW: A COMBINATION OF TWO MUTUALLY-SCRAMBLED LAGGED FIBONACCI GENERATORS

The lagged Fibonacci pseudorandom number generator has become increasingly popular due to its relative low cost, and the fact that it is simple to implement. Its general form is:

$$LF[r, s, m; \{x_0, \ldots, x_{r-1}\}],$$

where $r > s > 0$ are the lags of the past samples, m is the base, and $\{x_0, \ldots, x_{r-1}\}$ is a sequence of r initial values (seeds). For $n \geq r$ the sequence is characterized by a mapping of the type $x_n = x_{n-r} + x_{n-s}$, where $\{x_n\}$ denotes the output sequence and n denotes the time.

When $m = 2^N$, N being the word length, and the trinomial $x^r + x^s + 1$ is irreducible and primitive over GF(2), the maximum period p is reached (on condition that at least one of the seed values must be odd), and its value is $p = 2^{N-1}(2^r - 1)$.

However, LFGs have some limitations referred to randomness and security. On one hand, they fail to pass certain randomness tests. For instance, the LFGs *ran3*, *Ranlux*, *ranlxs0*, and *zuf*, of the GSL-GNU RNGs library [16], fail to pass the Birthday Spacing test of the Marsaglia's Diehard test suite [17]. On the other hand, LFGs suffer from a lack of security, because a simple mathematical analysis of the sequence of past generated numbers allows to predict the subsequent numbers [18]. Furthermore, in order to have a perfect random behavior and sufficiently large periods, the LFGs must have large lags but, due to its state being proportional to the lag, this will in turn imply the need of a large amount of memory [19], making them unsuitable for applications with limited resources such as those used in the IoT.

A. ARROW: A SECURE PRNG FOR THE IoT

To avoid the above mentioned problems a more elaborated architecture, named Arrow, is proposed. Arrow PRNG structure is depicted in Fig. 1. It is based on the combination,

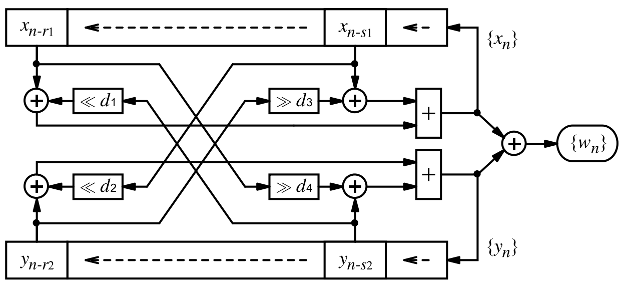


FIGURE 1. Block diagram of Arrow, composed of a combination of two mutually perturbed lagged Fibonacci generators.

by means of the bitwise XOR, of the outputs of two simple lagged Fibonacci generators.

Both LFGs are scrambled through the mutual perturbation of the most and least significant bits of the lagged samples, before the sum of their outputs mod m:

$$x_n = \left(x_{n-r_1} \oplus (y_{n-s_2} \ll d_1) + x_{n-s_1} \oplus (y_{n-r_2} \gg d_3)\right) \mod m,$$

$$y_n = \left(y_{n-r_2} \oplus (x_{n-s_1} \ll d_2) + y_{n-s_2} \oplus (x_{n-r_1} \gg d_4)\right) \mod m,$$

where d_1, d_2, d_3, d_4 are four constants, $0 < d_i < N$; $\oplus$ is the bitwise exclusive-or; $\gg$ and $\ll$ are the right-shift and left-shift operators in the C/C++ language. Note that $\gg d_i$ is equivalent to a multiplication by 2^{-d_i} followed by a floor operation, while $\ll d_i$ is equivalent to a multiplication by 2^{d_i} followed by a $\mod m$ operation. The output sequence of Arrow $\{w_n\}$ is:

$$w_n = x_n \oplus y_n, \tag{1}$$

were w_n is the output sample of the Arrow generator at the moment n and x_n, y_n are the output samples of the two generators, respectively.

The lags r_1, r_2 of both generators should be chosen of different values to ensure that the sequences from each generator have different lengths, in order to magnify the length of the final sequence. Moreover, the lags r_1, r_2, s_1, s_2 must be selected to satisfy that both trinomials $x^{r_1} + x^{s_1} + 1$ and $y^{r_2} + y^{s_2} + 1$ are irreducible and primitive over GF(2).

This structure improves the generator's security by increasing the number of states of the system, with the consequent increase of the period, the entropy, and the key space. It uses a mixing of arithmetical operations and bit-oriented operations to avoid purely algebraic and purely bit-oriented attacks. Additionally, when mixing multiple streams so that the size of the output word is smaller than the sum of the sizes of each input word, it is extremely hard to make an individualized analysis of the sequences generated by each generator; hence, it is more difficult to implement a cryptanalytic attack. Furthermore, only very efficient operations of easy implementation in hardware or software are used in the proposed structure: addition module the word size of the microcontroller, bitwise Boolean operations, and displacements of bits.

It was found that the mutual cross perturbation of the least significant bits of samples x_{n-r_1} and y_{n-r_2}, and the most significant bits of the samples x_{n-s_1} and y_{n-s_2}, before their addition, allows the generator to successfully pass previously-failed randomness tests. Specifically, the best solution was the cross perturbation achieved with only half the bit size of the samples, $d_i = N/2$. The sample x_{n-r_1} is right-shifted d_4 bits and then it is combined with y_{n-s_2} by means of a bitwise exclusive-or, while the sample x_{n-s_1} is left-shifted d_3 bits and then it is combined with y_{n-r_2}, also using a bitwise exclusive-or. The samples y_{n-r_2} and y_{n-s_2} are perturbed similarly. The opposite may also be used.

The seed of the generator is a sequence of $r_1 + r_2$ initial values $\{x_0, \ldots, x_{r_1-1}\}$ and $\{y_0, \ldots, y_{r_2-1}\}$. These values may be composed by the key values and the initialization vector (IV). The only forbidden seed is the null seed, which produces a null sequence.

An exhaustive search of the generator periods for several trinomials was carried out, and it was found that the mutual cross perturbation drastically increases the period of the generator. Indeed, the periods of Arrow are much larger than the periods of an equivalent generator built by the combination of two conventional LFGs using the bitwise XOR addition, without scrambling. In this last case the resulting period is the least common multiple of the periods of both generators. The maximum possible period of the Arrow generator corresponds to the amount of all possible states of the generator except the all-zeros state: $p = m^{r_1+r_2} - 1$.

By increasing the word length, the periods of Arrow grow much faster than the periods of the two conventional LFGs combined by the bitwise XOR addition. The longer periods happened for primitive trinomials. From the experimental data it can be concluded that the period lengths of any Arrow are greater than the square of the period length of the corresponding combined conventional LFG. The period of Arrow depends on the parameters $r_1, s_1, r_2, s_2, N, d_1, d_2, d_3, d_4$, and also on the initial values $\{x_0, \ldots, x_{r_1-1}\}$ and $\{y_0, \ldots, y_{r_2-1}\}$.

B. LINEAR COMPLEXITY OF ARROW

The *linear complexity* of a sequence is defined as the length of the shortest linear feedback shift register that generates it. It is a measure of the unpredictability of a random or pseudorandom sequence, being maximal when it is very close to half of its length [21]. This is determined through the Berlekamp-Massey LFSR synthesis algorithm [22].

The linear complexity of the least significant bits of the words of a conventional LFG is equal to the value of the lag r; the linear complexity of the successive significant bits increases progressively, reaching a value close to $2^{N-1}(2^r - 1)$ for the most significant bits. If two conventional LFGs, with lags r_1 and r_2, are combined by a bitwise XOR addition, the linear complexity of the bits of the resulting sequence is nearly equal to the sum of the respective complexities of the bits of both LFGs. This fact allows the easy determination of several of the least significant bits of the seed, thus simplifying a brute-force attack for the complete determination of the seed.

In the case of Arrow, the linear complexity of any level of significance is equal to half of the period of the sequence. Hence, the analysis of the linear complexity of any bit of the sequence is useless for guessing the seed.

One important property of a random sequence is the distance of occurrence between samples of equal value. The most probable distance between two identical samples of an ideal sequence is zero; when the distance between identical samples increases, the probability of coincidence between the two samples diminishes following the Poisson distribution.

Fig. 2 illustrates the number of identical samples in a LFG sequence with a given distance between them. It corresponds to a LFG sequence of 2^{20} samples, with $N = 8$, $r = 7$, $s = 3$. As a reference, the case of a perfect random sequence is also illustrated (in green). It can be seen that the set of samples of the LFG presents an irregular distribution, very far from the ideal one.

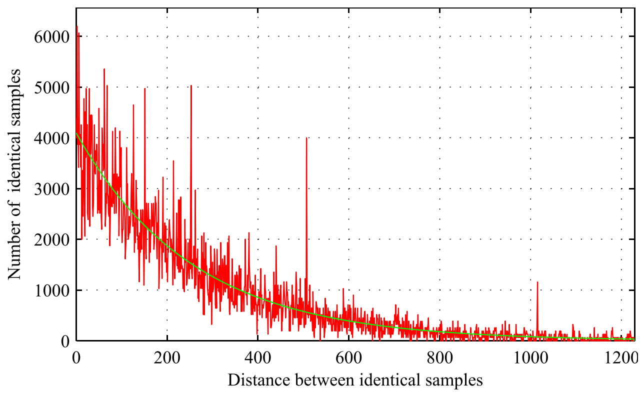


FIGURE 2. **Distribution of samples with equal value as a function of their distance: LFG (red) and a perfect random sequence (green).**

Fig. 3 corresponds to an Arrow sequence of 2^{20} samples, with $N = 8$, $r_1 = 7$, $s_1 = 3$, $r_2 = 3$, $s_2 = 1$. The distribution of samples is close to the ideal one.

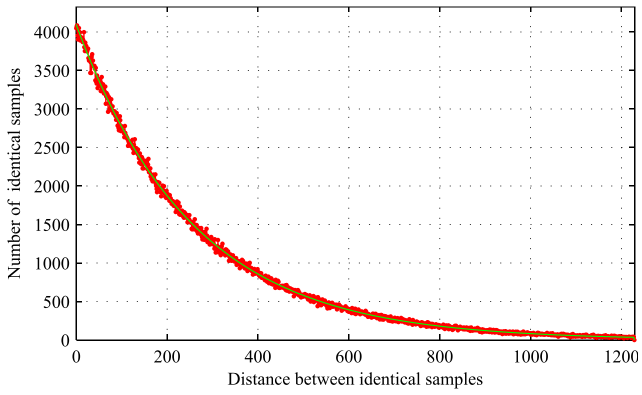


FIGURE 3. **Distribution of samples with equal value as a function of their distance: Arrow (red) and a perfect random sequence (green).**

The sequences generated by Arrow pass successfully all the Marsaglia's Diehard randomness test suite, as opposed to the conventional LFG that fails to pass some of them. Furthermore, Arrow also passes the randomness tests of NIST [20].

IV. IMPLEMENTATION AND RESULTS

The architecture of Arrow can be built with simple hardware or software. Additionally, it is flexible enough to allow an assortment of implementations with different word length on different platforms. The Arrow algorithm was implemented using C++ to first verify its correct operation. It is worth mentioning the good performance of Arrow on 64-bit processors, reaching 0.87 cycles/bit when programmed in C over a 2.93 GHz Intel Core i7 CPU 870.

The chosen design is based on microcontrollers due to the evident advantages for IoT applications such as reduced cost and size, high flexibility and, what is most important, that this design does not constrain the system to an specific board in the future. The Arduino platforms have been chosen over similar alternatives due to their lower price, strong support network, and extensive use in IoT designs [23]–[30]. Additionally, they have the advantages of being easy to build, and easy to update. In particular they have the potential benefit of uploading a new PRNG algorithm to upgrade the device, or even porting the software to another device with a different architecture.

The Arrow pseudorandom generator has been implemented on two different platforms: an Arduino UNO and an Arduino 101. The former is based on the ATmega328P microcontroller, a low-power 8-bit microcontroller, with 32 kB flash memory, a RAM of 2 kB, and 16 MHz clock frequency. The latter is based on a 32-bit Intel Quark$^{\text{TM}}$ microcontroller, an Intel Curie$^{\text{TM}}$ module with 384 kB flash memory (196 kB handy to Arduino sketch), and 24 kB SRAM available to Arduino sketch. Both boards were powered through a USB connection.

We used a C++ implementation of the generator on both platforms and it was compiled using Arduino IDE 1.6.13. The compiled programs were then directly flashed into each device. Cycle counts, code sizes and RAM usage were obtained using the Arduino IDE. Due to the small RAM size of the boards, the two irreducible primitive trinomials of the Arrow were selected as $x^{17} + x^3 + 1$ and $y^{31} + y^3 + 1$, since they have relatively low lag values of $r_1 = 17$ and $r_2 = 31$, respectively. Both implementations generated a sufficiently-long random sequence with $N = 8$ and $N = 32$ bits for Arduino UNO and Arduino 101, respectively. Consequently, the values of the shifts d_1, d_2, d_3, d_4 must satisfy $0 < d_i < 8$, or $0 < d_i < 32$, according to the used platforms.

For Arduino UNO, the seeding of the Arrow generator was performed in the same way as that of the simple LFGs, by filling in the trinomial registers with the key and the IV. The length of the key is 256 bits (32 bytes), and the length of the IV is 128 bits (16 bytes). The 32 bytes of the key are shared among both trinomials, in such a way that half of them are used to fill in register $\{x_0, \ldots, x_{15}\}$ and the other half to fill in register $\{y_0, \ldots, y_{15}\}$. The IV values fill in the remaining places: x_{16} is filled with 1 byte of the IV and $\{y_{16}, \ldots, y_{30}\}$

are filled with the remaining 15 bytes of the IV. For Arduino 101 the resulting key and IV lengths are 1024 and 512 bits, respectively.

Each microcontroller outputs a pseudorandom bit sequence. Each output was saved in a text file using a simple serial port terminal application.

A large number of sequences were generated with Arrow in the aforementioned platforms with a word size of 8 bits on Arduino UNO and 32 bits on Arduino 101, using different trinomials, as mentioned above. All of the generated sequences passed the randomness tests of NIST [20] and Diehard from Marsaglia [17].

The repetition periods of each non perturbed LFG of Arrow are $p_{\text{LFG},1} \simeq 16.78 \times 10^6$ bytes and $p_{\text{LFG},2} \simeq 2.75 \times 10^{11}$ bytes, respectively. When combined by a simple bitwise XOR addition, the resulting repetition period will be the least common multiple of both periods, approximately $p_{\text{LFG},1+2} \simeq 3.57 \times 10^{16}$ bytes. However, it is impossible to calculate the resulting repetition period of the two mutually perturbed combined LFGs but indeed it will be much greater. The estimated period is comprised between $p = 1.2 \times 10^{33}$ and $p = 4 \times 10^{115}$. The length of this sequence is widely sufficient for any cryptographical task; consider, for instance, that the maximum allowable length of the pseudorandom sequence generated with an AES, in counter mode, is 6.87×10^{10} bytes.

No attack faster than a brute force key search has been identified. Since the smallest key used is 256 bits long, such attack is unfeasible. Moreover, the structure of Arrow reasonably prevents an algebraic attack due to the impossibility of learning the internal state of the generator.

As a proof of concept we have implemented Arrow in software to provide the 16-bit random number required in the EPC C1G2 standard for RFID [31], using the Arduino platforms. Key length, throughput (assuming that clock frequency is 100 KHz), and footprint of Arrow are presented in Table 1. The throughput range achieved is acceptable, since RFID provides slow data rates as compared to other digital communication systems.

TABLE 1. Arrow performance in the used microcontroller platforms.

Arduino	Key	Footprint	Throughput
UNO	256 bits	1.968 bytes	6.25 kb/s
101	1024 bits	18.312 bytes	32 kb/s

Arrow has 256-bit and 1024-bit security levels, respectively. Lightweight Cryptography (LWC) Standard developed in ISO/IEC 29192 Part 1 establishes that 80-bit security is the minimum security strength needed for LWC (2016 LWC workshop by NIST), hence Arrow fulfils the general security requirements of the IoT.

A. HARDWARE COMPLEXITY ESTIMATION

The estimated hardware complexity of Arrow using a word size of 8 bits and 16 bits, for UMC 90 nm and ON Semiconductor (formerly AMIS) 0, 35 μm CMOS technologies, has been computed following [13]; results are presented in Table 2 and Table 3, respectively.

TABLE 2. GE estimation for 8-bit Arrow.

Trinomials $x^7 + x^3 + 1$; $x^9 + x^4 + 1$			
Operator	Num	UMC 90 nm	ONS $0,35\ \mu$m
XOR	24	59.52	55.92
ADD	16	119.2	101.44
Register	128	584.96	853.76
Control	≈ 20%	152.73	202.22
Total GE		916.41	1213.34

TABLE 3. GE estimation for 16-bit Arrow.

Trinomials $x^5 + x^2 + 1$; $x^3 + x + 1$			
Operator	Num	UMC 90 nm	ONS $0,35\ \mu$m
XOR	48	119.04	111.84
ADD	32	232.4	202.88
Register	128	584.96	853.76
Control	≈ 20%	187,28	233,69
Total GE		1123.68	1402.17

An implementation specially suited for RFID 16-bit random number generation using the trinomials of Table 2, will consist of two Scrambled Lagged Fibonacci Generators (SLFG) with lengths of 9 and 7 words, respectively, and word length $N = 8$ bits. The value of the perturbation is $d = 4$. The length of the key is 96 bits with a 32-bit IV. If the IV is considered a secret parameter, the resulting key length is 128 bits and the state space is $2^{128} - 1$. Both key lengths, 96 and 128 bits, meet the security requirements of the LWC standard.

Similarly, using the trinomials of Table 3, an implementation for RFID 16-bit random number generation will consist of two SLFG with lengths of 5 and 3 words, respectively, being $N = 16$ bits. The value of the perturbation is $d = 4$. The resulting key length is 128 bits and the state space is $2^{128} - 1$.

Table 2 and Table 3 show that the required amount of GE is considerably smaller than the maximum commonly assumed in the literature.

Table 4 presents a comparison of Arrow with other proposed PRNGs in terms of the key/IV size and the hardware complexity (GE). Note that Arrow is the only PRNG with a key size that meets the security requirements of the LWC standard.

TABLE 4. Comparison of Arrow with other PRNG.

PRNG	Key/IV size	GE
LAMED [11]	32/32	1585
Warbler[9]	60/32	1238
Melia-Segui[15]	16	761
J3Gen[14]	64	1419
Arrow 8bit	**128**	**1213**
Arrow 16bit	**128**	**1402**

V. CONCLUSION

A lightweight, fast, and cryptographically secure pseudorandom number generator useful for resource-constrained devices like those used in the IoT has been described. It is based on the combination of two mutually scrambled lagged Fibonacci generators. Some criteria for choosing parameters of the generator are proposed to offer a high level of security. The generated sequence passes successfully the most stringent randomness test suites. The algorithm was programmed in C++ with word sizes of 8 bits and 32 bits, using only fast operations: addition, bitwise XOR and right and left shift. The estimated hardware complexity using CMOS technology meets the established requirement to be $\leq$4000 GE. The good performance reached on low-cost microcontroller platforms makes the proposed PRNG suitable for the IoT applications, which use resource-constrained devices such as RFID tags and wireless sensor nodes.

REFERENCES

[1] D. Airehrour, J. Gutierrez, and S. K. Ray, "Secure routing for Internet of Things: A survey," *J. Netw. Comput. Appl.*, vol. 66, pp. 198–213, May 2016.

[2] N. Mouha. (2015). *The Design Space of Lightweight Cryptography*. NIST Lightweight Cryptography Workshop. [Online]. Available: https://hal.inria.fr/hal-01241013

[3] V. Fischer, "A closer look at security in random number generators design," in *Constructive Side-Channel Analysis and Secure Design* (Lecture Notes in Computer Science), vol. 7275. Berlin, Germany: Springer, 2012, pp. 167–182.

[4] A. Peinado, J. Munilla, and A. Fúster-Sabater, "EPCGen2 pseudorandom number generators: Analysis of J3Gen," *Sensors*, vol. 14, no. 4, pp. 6500–6515, 2014.

[5] A. B. Orúe, L. Hernández, and F. Montoya, "Trifork, a new pseudorandom number generator based on lagged Fibonacci maps," *J. Comput. Sci. Eng.*, vol. 2, no. 2, pp. 46–51, 2010.

[6] A. Francillon and C. Castelluccia, "TinyRNG: A cryptographic random number generator for wireless sensors network nodes," in *Proc. IEEE 5th Int. Symp. Modeling Optim. Mobile, Ad Hoc Wireless Netw.*, Apr. 2007, pp. 1–7.

[7] G. Lo Re, E. Milazzo, and M. Ortolani, "Secure random number generation in wireless sensor networks," in *Proc. ACM 4th Int. Conf. Secur. Inf. Netw. (SIN)*, 2011, pp. 175–182.

[8] V. Gaglio, A. De Paola, M. Ortolani, and G. Lo Re, "A TRNG exploiting multi-source physical data," in *Proc. ACM 6th Workshop QoS Secur. Wireless Mobile Netw. (Q2SWinet)*, 2010, pp. 82–89.

[9] K. Mandal, X. Fan, and G. Gong, "Design and implementation of Warbler family of lightweight pseudorandom number generators for smart devices," *ACM Trans. Embedded Comput. Syst.*, vol. 15, no. 1, pp. 1–28, 2016.

[10] J. Mabin, G. Sekar, and R. Balasubramanian, "Distinguishing attacks on (ultra-) lightweight WG ciphers," in *Lightweight Cryptography for Security and Privacy* (Lecture Notes in Computer Science), vol. 10098. Cham, Switzerland: Springer, 2017, pp. 45–59.

[11] P. Peris-López, J. C. Hernández-Castro, J. M. Estévez-Tapiador, and A. Ribagorda, "LAMED—A PRNG for EPC class-1 generation-2 RFID specification," *Comput. Standards Interfaces*, vol. 31, no. 1, pp. 88–97, 2009.

[12] O. Markku-Juhani and D. E. Saarinen, "A do-it-all-cipher for RFID: Design requirements (extended abstract)," Cryptology ePrint Archive, Tech. Rep., 2012/317, 2012. [Online]. Available: https://eprint.iacr.org/2012/317

[13] H. Martín, P. Peris-Lopez, J. E. Tapiador, and E. S. Millan, "An estimator for the ASIC footprint area of lightweight cryptographic algorithms," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1216–1225, May 2014.

[14] J. Melià-Seguí, J. Garcia-Alfaro, J. Herrera-Joancomartí, "J3Gen: A PRNG for low-cost passive RFID," *Sensors*, vol. 13, no. 3, pp. 3816–3830, 2013.

[15] J. Melià-Seguí, J. Garcia-Alfaro, and J. Herrera-Joancomartí, "Multiple-polynomial LFSR based pseudorandom number generator for EPC Gen2 RFID tags," in *Proc. 37th Annu. Conf. IEEE Ind. Electron. Soc. (IECON)*, Nov. 2011, pp. 3820–3825.

[16] GNU Scientific Library (GSL). *Random Number Generation*. Accessed: Oct. 2, 2017. [Online]. Available: https://www.gnu.org/software/gsl/doc/html/rng.html

[17] G. Marsaglia, "The Marsaglia random number CDROM including the Diehard battery of tests of randomness," Dept. Statist., Florida State University, Tallahassee, FL, USA., 1995. [Online]. Available: https://web.archive.org/web/20160125103112/ and http://stat.fsu.edu/pub/diehard/

[18] R. Anderson,"On Fibonacci keystream generators," in *Proc. 2nd Int. Workshop Fast Softw. Encryption*, 1995, pp. 14–16.

[19] P. Grassberger, "On correlations in 'good' random number generators," *Phys. Lett. A*, vol. 181, no. 1, pp. 43–46, 1993.

[20] L. E. Bassham, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. 800-22 Rev 1a, 2010.

[21] R. A. Rueppel, "Linear complexity and random sequences," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 219. Berlin, Germany: Springer, 1986, pp. 167–188.

[22] J. L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Inf. Theory*, vol. IT-15, no. 1, pp. 122–127, Jan. 1969.

[23] M. S. Perez and E. Carrera, "Time synchronization in Arduino-based wireless sensor networks," *IEEE Latin Amer. Trans.*, vol. 13, no. 2, pp. 455–461, Feb. 2015.

[24] A. Solano, R. Dormido, N. Duro, and J. M. Sánchez, "A self-provisioning mechanism in OpenStack for IoT devices," *Sensors*, vol. 16, no. 8, p. 1306, 2016.

[25] F. Salamone, L. Belussi, L. Danza, M. Ghellere, and I. Meroni, "An open source 'smart lamp' for the optimization of plant systems and thermal comfort of offices," *Sensors*, vol. 16, no. 3, p. 338, 2016.

[26] A. Di Nisio, T. Di Noia, C. G. C. Carducci, and M. Spadavecchia, "High dynamic range power consumption measurement in microcontroller-based applications," *IEEE Trans. Instrum. Meas.*, vol. 65, no. 9, pp. 1968–1976, Sep. 2016.

[27] P. Kosobutskyy and R. Ferens, "Statistical analysis of noise measurement system based on accelerometer-gyroscope GY-521 and Arduino platform," in *Proc. 14th Int. Conf. Exper. Designing Appl. CAD Syst. Microelectronics (CADSM)*, Feb. 2017, pp. 405–407.

[28] M. Kim, K. Kim, K. Seo, J. Lee, K. Park, and K. Kim, "Modeling process-aware Internet of Things services over an ARDUINO community computing environment," in *Proc. 19th Int. Conf. Adv. Commun. Technol. (ICACT)*, Feb. 2017, pp. 163–166.

[29] J. Avery, T. Dowrick, M. Faulkner, N. Goren, and D. Holder, "A versatile and reproducible multi-frequency electrical impedance tomography system," *Sensors*, vol. 17, no. 2, p. 280, 2017.

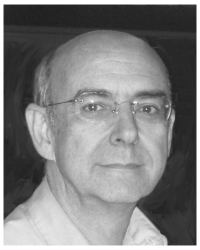
[30] A. M. Reyes, S. Herrera, M. A. Márquez, A. J. Calderón, I. González, and J. M. Andújar, "Easy handling of sensors and actuators over TCP/IP networks by open source hardware/software," *Sensors*, vol. 17, no. 1, p. 946, 2017.

[31] EPC Global. (2015). *UHF Air Interface Protocol Standard Generation2/V2.0.1*. Accessed: Sep. 27, 2017. [Online]. Available: http://www.gs1.org/epcrfid/epc-rfid-uhf-air-interface-protocol/latest

AMALIA BEATRIZ ORÚE LÓPEZ received the B.E. and M.S. degrees in telecommunications systems from the University of Oriente, Cuba, in 1984 and 1998, respectively, and the Ph.D. degree from the Polytechnic University of Madrid, Spain, in 2013. She is a Researcher with the Department of Information and Communications Technologies, Institute of Physical and Information Technologies, Spanish National Research Council, Madrid. She has authored several research papers in international scientific journals, peer-reviewed workshops, and conferences. She has served as a Referee for different SCI journals and international conferences. Her research interests comprise cryptography, Internet of Things security, information security, teaching methods of cryptography and educational innovation. She is a member of the IEEE Education Society and the IEEE Council on RFID.

LUIS HERNÁNDEZ ENCINAS received the Degree and the Ph.D. degree in mathematics from the University of Salamanca, Spain, in 1980 and 1992, respectively. He is currently a Researcher with the Department of Information and Communications Technologies, Institute of Physical and Information Technologies, Spanish National Research Council, Madrid, Spain. He has participated over 30 research projects. He has authored nine books, nine patents, over 150 papers, over 100 contributions to workshops, and conferences. He has supervised three doctoral thesis and has served as a referee for different SCI journals and for many international conferences. His current research interests include cryptography and cryptanalysis of public key cryptosystems (RSA, ElGamal, and Chor-Rivest), cryptosystems based on elliptic and hyperelliptic curves, graphic cryptography, pseudorandom number generators, digital signature schemes, authentication and identification protocols, crypto-biometry, secret sharing protocols, side channel attacks, and number theory problems.

AGUSTÍN MARTÍN MUÑOZ received the Degree and the Ph.D. degree in physics from the Complutense University of Madrid, Spain, in 1988 and 1995, respectively. He is currently a Researcher with the Department of Information and Communications Technologies, Institute of Physical and Information Technologies, Spanish National Research Council, Madrid. He has authored 20 research papers in international scientific journals and co-authored a book and over 60 contributions to peer-reviewed workshops, and conferences. He has participated in 20 research projects and contracts, has taught several courses and seminars, and has co-authored three patents. He started his research career dealing with the development of numerical methods in electromagnetics to study the interaction of electromagnetic radiation, especially in the radiofrequency range, with different objects as satellites or biological tissues. His latest research interests are focused on the analysis of possible vulnerabilities of radiofrequency identification systems and the study of techniques of physical attacks to cryptographic devices. He is member of the International Association for Cryptologic Research.



FAUSTO MONTOYA VITINI received the Ph.D. degree in telecommunication engineering from the Polytechnic University of Madrid, in 1971. He is currently a Professor Ad Honorem with the Department of Information and Communications Technologies, Institute of Physical and Information Technologies, Spanish National Research Council, Madrid, Spain. He has authored several research papers in international scientific journals, peer-reviewed workshops, and conferences. He has participated as a project leader of several international projects, Spanish Government projects and industry projects. He has authored nine international patents and 14 Spanish patents. He has supervised four doctoral thesis. His current research interests include cryptography, information security, watermarks, number theory, pseudorandom generators, non-linear dynamical systems and power ultrasonics.