Trifork, a New Pseudorandom Number Generator Based on Lagged Fibonacci Maps

A. B. Orue, F. Montoya, and L. Hernández Encinas

Abstract— A new family of cryptographically secure pseudorandom number generators, is described. It is based on the combination of the sequences generated by three coupled Lagged Fibonacci generators, mutually perturbed. The mutual perturbation method consists of the bitwise XOR cross-addition of the output of each generator with the right-shifted output of the nearby generator. The proposed generator has better entropy and much longer repetition period than the conventional Lagged Fibonacci Generator. It passed successfully the most stringent randomness test suites. The effective speed of generation is approximately of one bit per computer clock cycle. The algorithm was programmed in C99 with 64 bits of word size.

Index Terms—Pseudo random number generator, Lagged Fibonacci map.

1 Introduction

PSEUDO Random Number Generators (PRNG) are an essential part of any cryptosystem because of the security of many cryptographic systems depends on the generation of good pseudorandom sequences. The generated numbers will be used mainly as keystreams, initial vectors, private keys, and private signatures, destined to control or initialize cryptographic algorithms. However, the design of reliable pseudorandom generators remains an open problem in cryptology. Some de facto standards that regarded as secure in the past have recently failed [1], [2], [3] and [4], other generators such as the BBS generator —which is one of the few PRNGs with proven security [5]—, are of little use for its slowness.

In 2000, the NESSIE (New European Schemes for Signatures, Integrity and Encryption) project was launched in Europe, as an open call, for the submission of cryptographic primitives. Unfortunately, all six stream ciphers submitted failed against cryptanalysis. In 2004, the eSTREAM project was launched as part of ECRYPT (European Network of Excellence in Cryptology). eSTREAM issued a call for stream cipher primitives. As a result, in 2008 seven finalists where pre selected (four in software and 3 in hardware), but currently it has not yet been possible to decide which of them deserves to be a standard.

This paper presents a family of pseudo random number generators that consist of several coupled Lagged Fibonacci maps, mutually perturbed, that will serve as keystream in a stream cipher. The first implementation of this class of family of generators used three couple of sawtooth piecewise linear chaotic maps [6]. In this work we present a similar configuration but using couple three coupled Perturbed Lagged Fibonacci Generators.

- *A. B. Orue is with the Applied Physics Institute, Spanish National Research Council (CSIC), Madrid, Spain..*
- *F. Montoya is with the Applied Physics Institute, Spanish National Research Council (CSIC), Madrid, Spain.*
- *L. Hernández Encinas is with the Applied Physics Institute, Spanish National Research Council (CSIC), Madrid, Spain.*

2 Family of Pseudorandom Generators Based on the Combination of Perturbed Lagged Fibonacci Generators

The proposed family of pseudorandom generators is based on the combination of the sequences generated by several coupled basic pseudorandom generators, through a one-way function. Every generator has a limited number of states, and therefore its period of repetition is also limited; in agreement to the word length of the language with which it is programmed, that in turn depends on the word length of the hardware that is in use. The combination of several sequences by a one-way function has two aims. The first one is to increase the number of states of the system, with the consequent increase of the period of the repetition, the increase of the entropy, and the increase of the number of keys. The second objective is to increase the security. Indeed, when mixing multiple streams so that the size of the output word is less than the sum of the sizes of the input words, it is extremely difficult to make an individualized analysis of the sequences generated by each individual generator, hence, to mount a cryptanalytic attack.

2.1 Combination Method

The method of combination chosen consists of the bitwise XOR of the numbers generated by several simple generators. The number of generators must be chosen depending on the application and on the word length of the software with which it is programmed.

It was used a mixing of arithmetical operations and opera-

tions oriented to bit, because this serves to avoid the purely algebraic attacks and the purely bit oriented attacks. The mixing of a variety of operations, as algebraic and bit manipulations, prevents the mathematical behavior of the scheme from being shaped easily.

Furthermore, only very efficient operations are used: arithmetical operations (addition) module the word size of the compiler, bitwise Boolean operations, and displacements of bits, all of them of easy implementation in hardware or software. All these operations combined in the proposed generator contribute to a great mathematical complexity together with a high computational efficiency.

2.2 Lagged Fibonacci Pseudorandom Generator

In recent year Lagged Fibonacci pseudo-random number generators have become increasingly popular generators for serial as well as scalable parallel machines because it is easy to implement, it is cheap to compute and it does reasonably well on standard statistical tests [7], [8] and [9] especially when the lag is sufficiently high.

There is no other pseudo random generator simpler or faster than the Lagged Fibonacci, because it uses just one addition, while others use multiplications which are considerably more time consuming.

The Lagged Fibonacci Generators (LFG) have been widely studied [10], [11], [12] and [13]. The classic reference is [14]; Marsaglia [15] made a concise study for establishing the maximal periods, and choosing lags and starting values. A hardware accelerate version of the library SPRNG (Scalable Parallel Random Number Generator), consisting of six generators including two modified Lagged Fibonacci generators, are described in [16].

The general form of LFG is presented in [10], [11], [12] and [13] as:

$$LF\left[r,s,m,\circ;x_t\left\{0,\ldots r-1\right\}\right], \tag{1}$$

where $r>s>0$, are the lags, $\circ$ is a binary operation, m is the base and $x\left\{0,\ldots,r-1\right\}$ is a sequence of r initial values (seed). For $n\geq r$ the sequence is characterized by a mapping of the type,

$$x_n = x_{n-r}\circ x_{n-s}. \tag{2}$$

The usual operations $\circ$ are addition, subtraction, multiplication mod-m, and bitwise exclusive-or ($\oplus$) if m is a power of 2. Often $m=2^N$, where N is the word length of the machine.

The maximal repetition period is achieved when some conditions are satisfied by the initial values of the sequence and by the parameters r,s.

Goods candidates for values of r,s are primitive trinomials mod–2 [14]. There are several references where to find these trinomials [14], [17] and [18].

Lagged Fibonacci generators must be initialized at random, usually using another random number generator. Altman [19] indicates that initialization of the generator is a critical issue and pointed out that the bitwise random behavior of these generators depends on the generator used to initialize the LFG.

The properties of the addition and subtraction LFGs are basically the same. When $m=2^N$ and the trinomial x^r+x^s+1 is irreducible and primitive over GF(2), the maximal period p is reached, on condition that at least one seed must be odd [20], and its value is:

$$p=2^{N-1}(2^r-1), \tag{3}$$

but non primitive trinomial may lead to much shorter periods.

The LFG using the bitwise exclusive-or ($\oplus$) is known as the Tausworthe generator. When this one is coded with $N=1$, the linear feedback shift register (LFSR) generator is obtained.

2.3 Problems with Lagged Fibonacci Generators

The initialization of these generators is a very complex problem; any maximal period of generator has a large number of possible cycles, all different.

On the other hand the output of generator is very sensitive to initial conditions and statistical defects may appear initially but also periodically in the output sequence unless extreme care is taken.

Very few mathematical results have been derived about the randomness properties of these generators, making it necessary to rely on statistical tests rather than theoretical performance.

It is a well know fact that LFGs fails to pass certain randomness tests. For instance, all the LFGs *ran3*, *Ranlux* [21], *ranlxs0*, and *zuf*, of the the GSL-GNU RNGs library, fail to pass the Birthday Spacing test of the Marsaglia's Diehard test suite [22].

Most LFGs suffer from a fault of security, simple mathematical analysis of the sequence of past generated numbers allows for the prediction of next numbers, as the Siemen's Lagged Fibbonacci "FISH", which Ross Anderson showed that it can be broken with just a few thousand bits of known plaintexts [20].

2.4 Simple Perturbed Lagged Fibonacci Generator (PLFG)

The proposed family of pseudorandom generators is based on the combination of several simple Perturbed Lagged Fibonacci Generators (PLFG). Each generator consists of the modification of a conventional LFG perturbing the lower and higher bits of the samples, prior to their addition.

Assume that there are one-dimensional maps $LF\left[r,s,m,d,c_t\circ;x\left\{0,\ldots r-1\right\}\right]$ which consist of the modification of a Lagged Fibonacci Generator by means of the perturbation of the most and less significant bits of the samples, where $\left\{x_t\right\}$ denote the output sequence and r,s,m,d,c_t are the control parameters of the system, which is defined by the mapping:

$$x_n=\left(\left(x_{n-r}\oplus x'_{n-s}\right)+\left(x_{n-s}\oplus x'_{n-r}\right)\right)\bmod m, \tag{4}$$

$$x'_{n-s} = (x_{n-s} \gg d), \quad (5)$$

$$x'_{n-r} = (x_{n-r} \ll d). \quad (6)$$

where n denotes the time; m is the base $m = 2^N$; N is the word length; s and r are the lags of the past samples; d is a constant $2 \leq d \leq 0.7N$; $\oplus$ is the bitwise exclusive-or; $\gg$ and $\ll$ are the right-shift an left-shift operators in the C/C++ language, $\gg d$ is equivalent to a multiplication by 2^{-d} followed by a `floor` operation, while $\ll d$ is equivalent to a multiplication by 2^d followed by a $\bmod m$ operation.

The innovation of this generator is that it comprises three interlinked operations of different nature. The first one is an addition $\bmod 2^N$, peculiar to the algebraic pseudorandom generators, the second one is the bitwise exclusive-or, the third and four operations are the left-shift and right-shift, all three peculiar to the shift registers pseudorandom generators.

Different significant bits of the conventional LFG have different behaviors; for instance the full period of a conventional LFG, $2^{N-1}(2^r - 1)$ is attained only by the most significant bit if analyzed separately; if the bits are numbered from 1 (least significant bit) to N (the most significant bit), then bit k has period $2^{k-1}(2^r - 1)$ [23]. Is a well known fact that the behavior of least significant bit of the conventional LFG is the responsible of most randomness failures of the generator.

It was found that a good cure for the failures against stringent randomness tests was the perturbation of the low and high bits of the samples x_{n-s} and x_{n-r}, before their addition. The perturbation could be the bitwise exclusive-or of several bits of another sample x_{n-l}, but the use of a third sample to calculate the output sample x_n was expensive in time and resources. Hence the best solution was the cross perturbation of the two canonical samples. As described by (5) and (6), the sample x_{n-r} is left-shifted d bits and combined bitwise exclusive-or with x_{n-s}, while the sample x_{n-s} is right-shifted d bits and combined bitwise exclusive-or with x_{n-r}; the converse may be also used.

This perturbation increases dramatically the randomness and the period of the generator.

2.5 Experimental Results with Reduced Word Length and Lags

It was done an exhaustive search of the periods of the generator for various trinomials, with small lags and small values of *N*, a representative sample of the results is presented in Table 1, where *P* means the measured period of the perturbed generator. For this simple test the generator was initialized as a growing sequence of numbers $\{x_{n-k}\} = \{k\}$.

It was found that the periods of the PLFG are much greater than the periods of the conventional LFG, with the same lag and word length. When increasing the word length, the periods of the PLFG grew much faster than the periods of the conventional LFG.

The longer periods happened for primitive trinomials; non primitive trinomials, with the same r lag and word length, originated much smaller periods; but the period of the PLFG was always bigger than the period of the corresponding conventional LFG.

TABLE 1

REPETITION PERIODS OF TRINOMIALS

r	*s*	P/I *	*N*	*d*	*P*	*p*	*P/p*
3	1	I, P	3	2	102	28	3.64
			4	2	3 460	56	61.78
			5	3	29 131	112	260.10
			6	3	96 780	224	432.05
			7	4	1 864 621	448	4 162.10
			8	4	7 479 668	896	8 347.84
			9	4	12 808 196	1 792	7 147.43
4	1	I, P	3	2	3 375	60	56.25
			4	2	44 060	120	367.16
			5	3	874 496	240	3 643.73
			6	3	15 156 209	480	31 575.43
	2	I			336	192	1.75
5	2	I, P	3	2	30 766	124	248.11
			4	2	907 255	248	3 658.28
			5	3	> 2^{25}	496	> 67 650
	1	none			15 276 292	336	45 465.15
6	1	I, P	3	2	212 915	252	844.90
			4	2	12 798 758	504	25 394.36
7	1	I,P	3	2	1 122 624	508	2 209.89
			4	2	> 2^{25}	1 016	> 33 026
8	3	none	3	2	2 026 870	868	2 335.1
9	4	I, P	3	2	5 752 402	2044	2 814.28
	1	I			20 510 902	292	70 242.81
	3	none			255 816	84	3 045.42

* I = irreducible, P = primitive.

From the experimental data it can be concluded that the period lengths of any PLFGs is bigger than the square of the period length of the corresponding conventional LFG, for $N > 3$.

There is no apparent relation between the periods of the PLFG and the conventional LFG; a prime factorization of the period lengths of both generators showed that they share no more prime factors than it would be expected from completely random numbers.

Sometimes the measured period of the PLFG are primes, hence it is difficult to deduce a rule that can predict the period length by combining the parameter values in a similar way as expressed by (3).

It was also found that PLFGs with different d, but identical $r, s, N,$ and seeds have different periods.

Hence it can be resumed that the period of the PLFG depends on all four parameters $r, s, N,$ and d.

But oppositely, it was found that the measured period of the PLFG, when the trinomial is primitive, was independent of the set of selected seeds; as well as it happens with the conventional LFG. The only forbidden seed is $\{x_{n-k}\} = \{0\}$ which produces a null sequence.

The most important fact —and the fundamental goal of this research— was that the PLFG passed the entire Marsaglia's DIEHARD randomness test suite, which can be considered the most stringent one, at the present time, oppositely to the conventional LFG that fails to pass several tests.

The improvement of randomness and entropy of the PLFG can be appreciated on the return map, which is a plot of a time series as a function of the current and of the previous values.

The return map of the conventional LGH, is illustrated in Fig. 1, (with parameters $r = 3,\ s = 1,\ N = 6,$ after the generation of 8000 samples). The evident problem is that it is very sparse, it can be seen that to each value of x_n correspond very few different values of $x_{n+(r-s)}$.

There are $2^N \times 2^N = 1024$ possible combinations in the return map; but because of the short period of 112 samples, only a short fraction of all possible points of the return map are visited; they are visited repeatedly period after period. This characteristic denotes a lack of entropy and a high level of determinism, which can help for the cryptanalysis of the generator.

Fig. 2 illustrates return map of the PLFG, with parameters $r = 3,\ s = 1,\ N = 6,\ d = 3,$ after the generation of 8000 samples. It can be seen that the map is completely filled. All possible combinations of values of x_n and $x_{n+(r-s)}$ takes place. This characteristic denotes a high level of randomness, which obstructs the cryptanalysis of the generator.

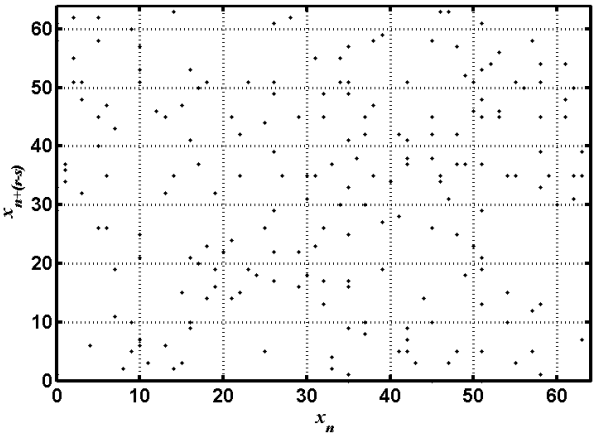

Fig. 1. Return map of the conventional LFG, *r* =3, *s*=1, *N*=6. Each point corresponds to the occurrence of a x_n and $x_{n+(r-s)}$ pair.

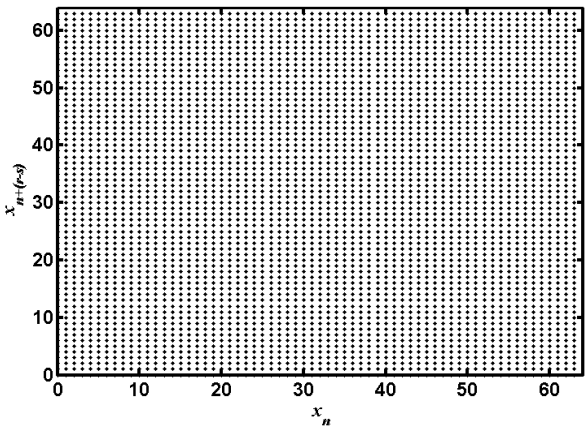

Fig. 2. Return map of the Return map of the PLFG, *r* =3, *s*=1, *N*=6 Each point corresponds to the occurrence of a x_n and $x_{n+(r-s)}$ pair.

2.6 Full Version of the Perturbed Lagged Fibonacci Generator

The full version of the PLFG was programmed in C99, with 64 bits of word length.

The most appropriate parameters r and s are the same used in the conventional LFG, which are the primitive irreducible trinomials over GF(2). They can be found tabulated in [18]; the easiest way of finding them is to select 2^r as a Mersenne prime, because in such case all irreducible trinomials are also primitive, they can by found in [24], some of them are:

$x^7 + x^1 + 1;\ x^{17} + x^3 + 1;\ x^{31} + x^3 + 1;\ x^{89} + x^{38} + 1;$
$x^{127} + x^1 + 1;\ x^{521} + x^{32} + 1;\ x^{607} + x^{105} + 1;$
$x^{1279} + x^{216} + 1;\ x^{2281} + x^{715} + 1;\ x^{3217} + x^{67} + 1.$

The seeding of the PLFG was done with a linear congruential generator, which generated the necessary seeds X_1 to X_r, as:

$$X_{k+1} = aX_k + c, \tag{7}$$

the first seed value X_1 is calculated from the generator key X_0, which is an arbitrary 64 bit number; a and c are two parameters, they were arbitrarily chosen as two 40 bit twin primes:

$$a = 1000000000061 \text{ and } c = 1000000000063.$$

This type of seeding guarantees a good beginning of the sequence, the election parameters a and c is not critical, but they should be big numbers relative to m. Suppose that $a = 1, c = 0$ and $X_0 = 1$, in such case a big quantity of numbers with very small values will lead the generated sequence, compromising its entropy, unless the first 1000 generated numbers were dropped.

The sequences generated by this version of the generator pass successfully the randomness test suites of the American

National Institute of Standards and Technology, NIST SP 800-22 [25], as well as the more stringent *Diehard* from Marsaglia [26].

3 TRIFORK: COMBINATION OF THREE PERTURBED LAGGED FIBONACCI GENERATORS

The full version of the generator defined by (4), (5) and (6), may constitute itself an excellent random number generator; but there is a remaining problem to be considered.

The problem is that the output generated numbers are the same that are used to calculate the next sample. An opponent may try to mount an algebraic attack, to determine the key of the generator, which could be feasible if the attacker have access to a large computer facility. To avoid this trouble, a more elaborated architecture, named *Trifork,* is proposed.

Trifork has three branches; each one formed by a Perturbed Lagged Fibonacci Generator. Two branches are combined by the bitwise XOR addition, to form the output of the joint generator; the third one will remain completely hidden. In this way the analysis of the output sequence is useless for determining the system parameters.

The three branches are interconnected by chained perturbing each other in a cyclic fashion. This architecture is depicted in Fig. 3.

It was found that, when combining by the bitwise XOR addition of two (or more) PLFGs, it was not necessary to implement the right-shift and left-shift operations (5) and (6), to separately perturb each lagged sample x_{n-r} and x_{n-s}, —which was mandatory in the case of an individual PLFG to pass with success all randomness tests—; instead, it was sufficient to cross perturb the input of each generator by the bitwise XOR addition of its output with the right-shifted output of the cyclic nearby PLFG.

Hence, the Trifork generator is defined as:

$$w_n = x_n \oplus z_n \tag{9}$$

$$x_n = \left(x_{n-r1} + x_{n-s1}\right) \bmod m \oplus z'_n, \tag{10}$$

$$y_n = \left(y_{n-r2} + y_{n-s2}\right) \bmod m \oplus x'_n, \tag{11}$$

$$z_n = \left(z_{n-r3} + z_{n-s3}\right) \bmod m \oplus y'_n, \tag{12}$$

$$x'_n = \left(\left(x_{n-r1} + x_{n-s1}\right) \bmod m\right) \gg d, \tag{13}$$

$$y'_n = \left(\left(y_{n-r2} + y_{n-s2}\right) \bmod m\right) \gg d, \tag{14}$$

$$z'_n = \left(\left(z_{n-r3} + z_{n-s3}\right) \bmod m\right) \gg d. \tag{15}$$

were w_n is the output sample of the Trifork generator at the moment n; x_n, y_n, z_n, are the output samples of the three generators; $r1, s1, r2, s2, r3, s3$, are the correspondent values of the lags; $r1, r2, r3$, should be chosen of different values, to warrant that the sequences generated by each generator have different lengths, in order to magnify the length of the final sequence.

The seeding of the Trifork generator is made in the same way as the simple PLFG, defined by the equations:

$$X_{k+1} = aX_k + c, \tag{16}$$

$$Y_{k+1} = aY_k + c, \tag{17}$$

$$Z_{k+1} = aZ_k + c, \tag{18}$$

were $\{X_k\}, \{Y_k\}, \{Z_k\}$, are the sets of seeds of each individual PLFG. The key of the Trifork is composed by X_0, Y_0 and Z_0, which are three arbitrary 64 bit numbers, hence the key has a length of 192 bits.

In this way two goals are attained: first, the key length of the generator is increased to the triple of one PLFG; second, the period of the joint generator is increased to the least common multiple of the periods of the three individual generators.

The evident form to attack the system is the brute force but the huge number of different keys prevents such attack. The algebraic attack is reasonably unlikely due to the impossibility of learning the internal state of the generator.

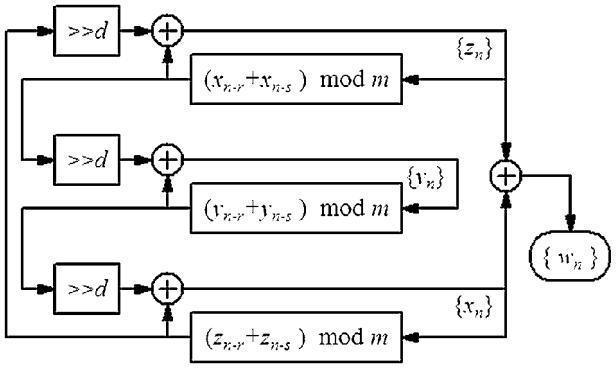

Fig. 3. Trifork combined generator.

Large number of sequences was generated by the Trifork combined generator with a word size of 64 bits, programmed in C99. All of them passed with success the randomness test suites of the NIST SP 800-22, as well as the *Diehard* from Marsaglia.

The performance of Trident in an Intel *Core2 Duo* with OS Windows 32 is about one clock cycle/bit. This speed is in the range of the finalists of the eSTREAM project.

Different versions may be designed using more than three coupled PLFG perturbed in the same way, but with different word sizes. For instance, to compensate the smaller periods attainable with architecture of only 32 bits, five coupled PLFG can be used, in this way two completely different sequences, with the same repetition period, could be generated.

4 Conclusion

A fast, cryptographically secure pseudorandom number generator has been described, based on the combination of three coupled Perturbed Lagged Fibonacci Generators. Its period is much longer than the conventional Lagged Fibonacci generator. Its output is unpredictable. The generated sequence passes successfully the most stringent randomness test suites. The algorithm was programmed in C99 with 64 bits of word size, using only fast operations: addition, bitwise XOR and right sift; hence, it was attained an excellent performance of about one clock cycle per generated bit.

Acknowledgment

This work has been partially supported by Ministerio de Industria, Turismo y Comercio (Spain) under grant TSI-020100-2009-44 and by Ministerio de Ciencia e Innovación (Spain) under grant TEC2009-13964-C04-02.

References

[1] A. Klein, "Attacks on the RC4 stream cipher," Designs, Codes and Cryptography, vol. 48, no. 3, pp. 269–286, 2008.Cryptography, vol. 48, no. 3, pp. 269–286, 2008.

[2] I. Goldberg and D. Wagner, "Randomness and the Netscape browser," Dr.Dobb's Journal, pp. 66–70, 1996.

[3] Z. Gutterman, B. Pinkas, T. Reinman, "Analysis of the Linux Random Number Generator". *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pp. 371–385 (2006)

[4] L. Dorrendorf, Z. Gutterman, B. Pinkas. "Cryptanalysis of the random number generator ofthe Windows operating system". *ACM T. Inform. System* vol. 13(1), pp. 10:1–10:32 (2009)

[5] L. Blum, M. Blum, and M. Shub, "A simple unpredictable pseudorandom number generator," *SIAM Journal on Computing*, vol. 15, pp. 364–383, 1986.

[6] A. B. Orúe, G. Alvarez, A. Guerra, G. Pastor, M. Romera, and F. Montoya, "Trident, a new pseudo random number generator based on coupled chaotic maps". *ArXiv*: 1008.2345, 2010.

[7] G. Marsaglia, "A current view of random number generators". *Computing Science and Statistics: Proceedings of the XVIth Symposium on the Interface*, pp. 3–10, 1985.

[8] G. Marsaglia, Diehard battery of tests of randomness, The Marsaglia random number CDROM, Department of Statistics, Florida State University, 1995.

[9] G. Marsaglia and W.W. Tsang, "Some difficult-to-pass tests of randomness", *Journal of Statistical Software,* Vol. 7, Issue 03, 2002.

[10] S. L. Anderson, "Random number generators on vector supercomputers and other advanced architectures," *SIAM Review*, vol. 32, pp. 221–251, 1990.

[11] J. Makino, "Lagged-Fibonacci random number generator on parallel computers," *Parallel Computing*, vol. 20, pp. 1357–1367, 1994.

[12] N. Masuda and F. Zimmermann, "PRNGlib: A parallel random number generator library," *Swiss Center for Scientific Computing* Technical Report: TR-96-08, 1996.

[13] Y. Bi, G. L. Warren, G. D. Peterson, R. J. Harrison, "A reconfigurable supercomputing library for accelerated parallel lagged-Fibonacci pseudorandom number generation". *Proceedings of the IEEE/ACM Conference on Supercomputing* (SC'06), 2006

[14] D. E. Knuth, *The Art of Computer Programming*, 3rd ed. Addison-Wesley, vol. 2, Seminumerical Algorithms, 1997.

[15] G. Marsaglia and L. H. Tsay, "Matrices and the structure of random number sequences," *Linear Alg. and Applic.*, vol. 67, pp. 147–156, 1985.

[16] J. Lee, G. D. Peterson, R. J. Harrison, and R. J. Hinde, "Implementation of Hardware-Accelerated Scalable Parallel Random Number Generators" *VLSI Design*, vol. 2010, Article ID 930821, 11 pages, 2010. doi:10.1155/2010/930821

[17] R. P. Brent, Fast and reliable random number generators for scientific computing, P*roc. PARA'04 Workshop on the State-of-the-Art in Scientific Computing*, pp. 1–10, 2004

[18] N. Zierler and J. Brillhart, "On primitive trinomials (mod 2)," *Information and Control*, vol. 13, pp. 541–554, 1968.

[19] N. S. Altman, "Bit-wise behavior of random number generators," *SIAM J. Sci. Stat. Comput.*, vol. 9, pp. 941–949, 1988.

[20] R. Anderson, "On Fibonacci keysteam generator", *Fast Software Encryption: Second International Workshop*. (FSE'94), Springer, pp. 346-352, 1994. (Conference proceeding)

[21] M. Luescher, "A portable high-quality random number generator for lattice field theory calculations". *Computer Physics Communications*, vol. 79, pp. 100–110, 1994.

[22] W.W. Tsang, C.W. Tso, L. Hui, K.P. Chow, K.H. Pun, C.F. Chong, H.W. Chan, "Development of Cryptographic Random Number Generators", *HKU CSIS Tech Report* TR-203-07, 2003, available at: http://www.csis.hku.hk/research/techreps/document/

[23] M. K. Chetry and W. B. V. Kandaswamy, "A Note On Self-Shrinking Lagged Fibonacci Generators", International *Journal of Network Security*, vol.11, no.1, pp. 11, 2010.

[24] N. Zierler, "Primitive trinomials whose degree is a Mersenne exponent", *Inform. and Control,* vol. 15, pp. 67-69, 1969.

[25] NIST800-22, National Institute of Standards and Technology, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications". *Special Publication*, SP 800-22Rev 1a, 2010.

[26] G. Marsaglia, "Diehard battery of tests of randomness", *The Marsaglia random number CDROM*, Department of Statistics, Florida State University, 1995, available at http://stat.fsu.edu/ ~geo/diehard.html.

Information about authors:

Amalia Beatriz Orúe López received the B.E. and Master of Science in Telecommunications systems, from University of Oriente, Cuba in 1984 and 1998 respectively, and obtained the Diploma of Advanced Studies from the Polytechnic University of Madrid, Spain in 2007. Her research interests comprise cryptography and chaotic cryptosystems.

Fausto Montoya obtained his Ph.D. in Telecommunication Engineering from the Polytechnic University of Madrid, in 1971. He is a researcher at the Department of Information Processing and Coding, Spanish Council for Scientific Research (CSIC). His current research interests include cryptography and chaotic dynamical systems.

Luis Hernández Encinas obtained his Ph.D. in Mathematics from the University of Salamanca, in 1992. He is a researcher at the Department of Information Processing and Coding, Spanish Council for Scientific Research (CSIC). His current research interests include cryptography, algebraic curve cryptosystems, image processing, and number theory.