Hindawi Publishing Corporation
Advances in Artificial Intelligence
Volume 2011, Article ID 464971, 10 pages
doi:10.1155/2011/464971

Research Article

Generalization of the Self-Shrinking Generator in the Galois Field GF(p^n)

Antoniya Todorova Tasheva,[1] Zhaneta Nikolova Tasheva,[2] and Aleksandar Petrov Milev[3]

[1] *Computer Systems Department, Faculty of Computer Systems and Control, Technical University of Sofia, 8 Kliment Ohridski Street, Sofia 1000, Bulgaria*
[2] *Communication and Computer Technique Department, National Military University "Vasil Levski", 1a Karel Shkorpil Street, Shumen 9701, Bulgaria*
[3] *Computer System and Technology Department, University of Shumen "Bishop Konstantin Preslavsky", 115 Universitetska Street, Shumen 9712, Bulgaria*

Correspondence should be addressed to Zhaneta Nikolova Tasheva, tashevi86@yahoo.com

Received 13 December 2010; Revised 5 February 2011; Accepted 24 February 2011

Academic Editor: Farouk Yalaoui

The proposed by Meier and Staffelbach Self-Shrinking Generator (SSG) which has efficient hardware implementation only with a single Linear Feedback Shift Register is suitable for low-cost and fast stream cipher applications. In this paper we generalize the idea of the SSG for arbitrary Galois Field GF(p^n). The proposed variant of the SSG is called the p-ary Generalized Self-Shrinking Generator (pGSSG). We suggest a method for transformation of a non-binary self-shrunken pGSSG sequence into balanced binary sequence. We prove that the keystreams of the pGSSG have large period and good statistical properties. The analysis of the experimental results shows that the pGSSG sequences have good randomness properties. We examine the complexity of exhaustive search and entropy attacks of the pGSSG. We show that the pGSSG is more secure than SSG and Modified SSG against these attacks. We prove that the complexity of the used pGSSG attacks increases with increasing the prime p. Previously mentioned properties give the reason to say that the pGSSG satisfy the basic security requirements for a stream chipper and can be useful as a part of modern stream ciphers.

1. Introduction

The binary Pseudorandom Sequences (PRSs) with maximum period and good statistical and correlation properties have established themselves as foundation for generation of many signals used in modern communication and information systems. Among the most important applications [1, 2] of these signals are the Code Division Multiple Access (CDMA) systems, wireless networks, and communication systems where the multiple access interference is minimized. Generators of pseudorandom sequences for security at authorization process and stream cipher are other key areas of the PRSs implementation in different kinds of local and global networks.

A great number of methods for generation of pseudorandom sequences are used in practice [3–13]. They are divided as linear and nonlinear [5]. Most linear methods such as Linear Congruent Generator, Multiplicative Linear Congruent Generator and Multiply Recursive Generator [5] cannot be used for ensuring the security and for information encryption in the communication network. Digital multistep and formal series [5] can solve that problem but they are too slow because many steps are done recursive. In order to ensure practical stability of a chosen crypto-algorithm it is necessary to break the linearity in the generated linear sequence. It could be done by applying a nonlinear function over a part of the generated bits for additional allocation [5, 12]. The binary sequences with maximum period (m-sequences) are mainly used in many communication systems because of their properties such as uniformly allocation of the binary digits and the runs over its period and the ideal two-level autocorrelation. The main methods for generation

of the m-sequences are Linear Feedback Shift Register (LFSR) which function is described by algebra in finite Galois field GF(2) and Feedback with Carry Shift Register (FCSR) where the algebra in finite 2-adic field is used.

Self-shrinking generator [11] is a keystream generator used as a stream cipher. It is based on the shrinking principle [4, 14] and has remarkably low hardware requirements. So far, it has shown considerable resistance against cryptanalysis. The binary sequences generated by shrinking generator have very good encryption properties [4, 5, 9, 12, 15, 16].

The self-shrinking generator (SSG) was proposed by Meier and Staffelbach at Euro-crypt'94 in [11]. It is a variant of the original Shrinking Generator given by Coppersmith et al. in [4]. This amazingly simple cipher, containing a single LFSR, has up to now surprisingly well-resisted all the known cryptographic attack techniques. In [17] Mihaljevic presented a faster attack with minimal time complexity $O(2^{0.5L})$ that needs a longer part of keystream sequence. It is shown that cryptanalysis is successful with high probability after 2^{L-l} steps, $l \leq L/2$, where l is l-bit section of the keystream and L is the key length. Since the procedure is done for $L/4 \leq l \leq L/2$, the running time can vary from $2^{0,5L}$ in the very best case to $2^{0,75L}$ under more unfavorable circumstances. Later Zenner et al. improve the cryptanalysis of self-shrinking generator. The attack [18] is based on a backtracking algorithm and will reconstruct the key from a short sequence of known keystream bits. The algorithm takes at most $O(2^{0.694L})$ steps, where L is the key length.

An attack on SSG requiring very small keystream data $(2.41n)$ is the binary decision diagram (BDD) cryptanalysis proposed by Krause in [19] with time complexity $O(2^{0.656n})$ and equivalent memory complexity $O(2^{0.656n})$. The BBD-attack was improved in [20] with the same time complexity and only $O(n^2)$ memory complexity.

The best tradeoff between time, memory and data complexity today is the new guess-and-determine cryptanalysis [21]. The time complexity of this attack is $(2^{0.571n})$ and memory complexity $O(n^2)$ from $(2^{0.194n})$ keystream bits for $n < 100$, and time complexity $O(2^{0.556n})$ and memory complexity $O(n^2)$ from $(2^{0.161n})$ keystream bits for $n \geq 100$.

In this paper a p-ary generalized self-shrinking generator which produces nonbinary sequences is proposed. A method for transformation of nonbinary self-shrunken sequence into balance binary sequence is given.

The paper is organized as follows. First, the basic principles of the p-ary LFSR are described. In Section 3 the function of the Self-Shrinking Generator is given. The working algorithm of the p-ary Generalized Self-Shrinking Generator for Galois Field GF(p^n) is given in Section 4. Then the properties of the proposed pGSSG are discussed and analyzed. Finally, the security of the pGSSG against the exhaustive search and entropy attacks are analysed.

2. Basics Principles of the p-Ary LFSR

The basic principles of wide spread LFSR registers [5, 7, 10, 12] for GF(p^n) when p is arbitrary prime will be given in this section.

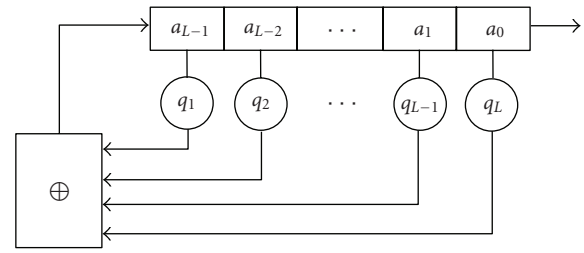

Figure 1: pLFSR register with Fibonachi architecture.

Real systems use pLSFR constructed in two different architectures Fibonachi and Galois. The properties of the finite Galois field [7, 22] are used.

Definition 1. If p is a prime then generated p-ary pseudorandom sequence (pPRS) consists of numbers $0 \div p-1$ and the appearance of every number is independent by the value of the previous and the following digits in the sequence.

2.1. pLFSR with Fibonachi Architecture

Definition 2. p-ary LFSR with Fibonachi architecture and length L (Figure 1) consists of L stages (delay elements) a_i, $i = 0, 1, \ldots, L-1$, and one input and one output.

Every element can remember one p-ary number. The register is initialized by p-ary sequence $(a_0, a_1, \ldots, a_{L-1})$.

During each clock cycle the following operations are performed.

(1) The content of stage 0 is output and forms part of the output sequence.

(2) The content of element i is passed to element $i-1$ for each i, $1 \leq i \leq L-1$.

(3) A calculation of the linear recurrent dependency is done

$$a_n = \sum_{i=1}^{L} q_i a_{n-i} \bmod p, \quad \text{for } n \geq L \tag{1}$$

and its result is stored in the most left element of the register.

The output sequence is described by (1) taking into consideration $n \geq L$, the first $L-1$ p-ary digits are eliminated from the output sequence.

Every configuration of this architecture is defined by the feedback coefficients q_i, $i = 1, 2, \ldots, L$. The generated output sequence $A = (a_0, a_1, a_2, \ldots)$ is well defined for $q_L \neq 0$ and the feedback polynomial

$$q(x) = q_L x^L + q_{L-1} x^{L-1} + \cdots + q_1 x - 1. \tag{2}$$

Every infinite p-ary sequence A can be represented by its generating function

$$A(x) = \sum_{i=0}^{\infty} a_i x^i \tag{3}$$

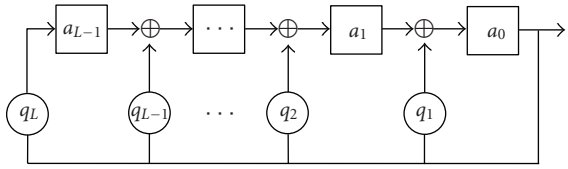

Figure 2: pLFSR register with Galois architecture.

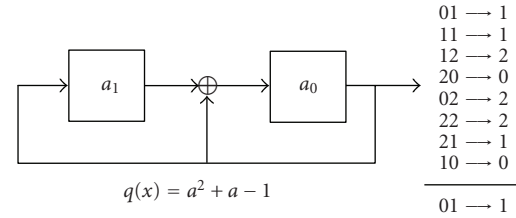

Figure 3: 3LFSR register with Galois architecture and length $L = 2$.

which is an element of the ring GF(p)[x] of the formal powers with coefficients in GF(p).

The generated sequence A is periodic only if the generating function $A(x)$ could be represented as a quotient of two polynomials

$$A(x) = -\frac{h(x)}{q(x)}. \tag{4}$$

The generated sequence A is strictly periodic if the following condition is in effect:

$$\deg(h(x)) < \deg(q(x)). \tag{5}$$

The reverse dependence is in effect as well. If $B = (b_0, b_1, b_2, \ldots)$ is a strictly periodic sequence and $B(x) = -h(x)/q(x)$ is its generating function then $q(x)$ is a connection polynomial of the pLFSR which generates sequence B and a polynomial $h(x)$ defines the initial state of the register by using

$$h(x) = \sum_{k=0}^{L-1} \sum_{i=0}^{k} q_i a_{k-i} x^k. \tag{6}$$

2.2. pLFSR with Galois Architecture

Definition 3. p-ary LFSR with Galois architecture and length L (Figure 2) consists of L delay elements a_i, $i = 0, 1, \ldots, L-1$, and one input and one output. The multipliers of the feedback are given as $q_1, q_2, \ldots, q_L$. Every element can remember one p-ary number. The register is initialized by p-ary sequence $(a_0, a_1, \ldots, a_{L-1})$.

During each clock cycle the following operations are performed.

(1) The content of stage 0 is output and forms part of the output sequence.

(2) The content of element i is passed to element $i-1$ for each i, $1 \le i \le L-1$, and the following recurrent dependencies are in effect

$$\begin{aligned} a_i' &= (a_{i+1} + q_{i+1} a_0) \bmod p, \quad 0 \le i \le L-2, \\ a_{L-1}' &= q_L a_0 \bmod p. \end{aligned} \tag{7}$$

(3) The output of the element 0 is implemented in every multiplier of feedbacks and a sum with the previous element by modulo p is done.

The Galois architecture of pLFSR is described by the feedback polynomial $q(x)$ which is defined by (2) and the polynomial $h(x)$

$$h(x) = a_0 + a_1 x + \cdots + a_{L-1} x^{L-1} \tag{8}$$

which is defined by the initial state of the register $(a_0, a_1, \ldots, a_{L-1})$.

The generated output sequence $B = (b_0, b_1, b_2, \ldots)$ is a sequence of polynomial coefficients $B(x) = -h(x)/q(x)$. The reverse statement for defining $q(x)$ and initial state by using (8) is also in effect.

2.3. Properties of the pLFSR Sequences. One of the most important properties of the pLFSR is the maximal period [22] and it depends on the length of the registers L

$$T \le p^L - 1. \tag{9}$$

pPRS sequences which have a maximal period (mp-sequences) are used in cryptography and in communication systems for designing complex signals due to their good correlation properties. In order to generate mp-sequences it is necessary that the feedback polynomial $q(x)$ of the pLFSR is primitive in extended Galois field GF(p^L).

The uniform allocation of the p-ary digits and their runs (series) is another important property. The following statements are true.

(i) The number of appearances of all nonzero elements N_j in maximal period $T = p^L - 1$ is $N_i = p^{L-1}$ for $i = 1, 2, \ldots, p-1$ and the number of zero elements is $N_0 = p^{L-1} - 1$.

(ii) Every nonzero p-ary runs (series) B with length s, $1 \le s \le L$ appears in a period p^{L-s} times and $p^{L-s} - 1$ times if it is zero.

Figure 3 shows the 3LFSR register with Galois architecture and length $L = 2$. As one can see the output numbers 1 and 2 appear $N_1 = N_2 = 3$ times in one period and the number of zeros is $N_0 = 2$. All nonzero runs with length 2 appear only ones and zero run is not used.

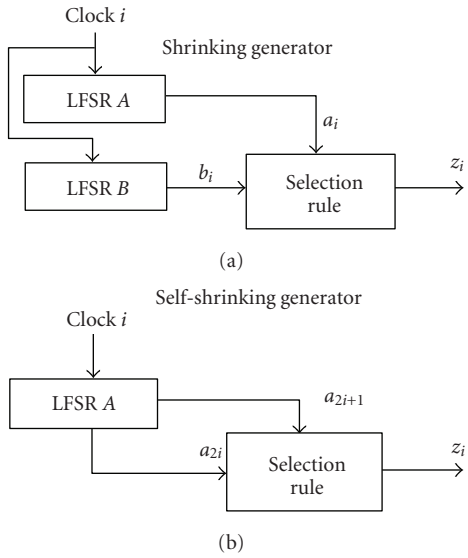

Figure 4: Shrinking and self-shrinking generator.

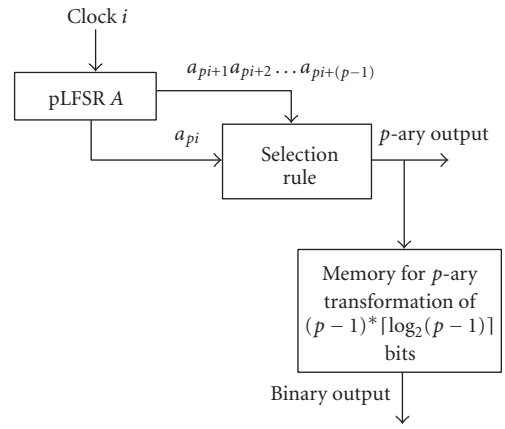

Figure 5: p-ary Generalized Self-Shrinking Geneator.

3. Nonbinary Self-Shrinking Generator

3.1. Basics of Shrinking Generators. The shrinking generator [4, 9] consists of two Linear Feedback Shift Registers (LFSR) A and B generating the m-sequences $(a_i)_{i\geq 0}$ (denoted as A sequence) and $(b_i)_{i\geq 0}$ (denoted as B sequence), respectively. The keystream sequence $(z_j)_{j\geq 0}$ is constructed from these two sequences according to the following selection rule: for every clock i, consider the selection bit b_i. If $b_i = 1$, the output is a_i. Otherwise, discard both b_i and a_i. This way, a nonlinear keystream is generated. Even a cryptanalyst who knows part of the keystream sequence cannot tell easily which $(z_j)_{j\geq 0}$ corresponds to which a_i, since the length of the gaps (i.e., the number of a_i that has been discarded) is unknown. In [4, 14] the shrinking generator is shown to have good algebraic and statistical properties. It is shown that the number of attacks that could reconstruct the initial state of A and B increases and requires exponential running time in the length of $|B|$ register (LFSR B).

The self-shrinking generator is a modified version of the shrinking generator and was first presented by Meier and Staffelbach in [11].

The self-shrinking generator requires only one LFSR A, whose length will be denoted by L. The LFSR generates an m-sequence $(a_i)_{i\geq 0}$. The selection rule is the same as for shrinking generator, using the even bits $a_0, a_2, a_4, \ldots$ as B-bits and the odd bits $a_1, a_3, a_5, \ldots$ as A-bits in the above sense. Thus, the self-shrinking rule requires a couple (a_{2i}, a_{2i+1}) as input and outputs a_{2i+1} if $a_{2i} = 1$.

The close relationship between shrinking and self-shrinking generator is shown in Figure 4.

In [12] an algorithm is given that transforms an L-bit self-shrinking generator into a $2L$-bit shrinking generator. It is also shown that a shrinking generator with register lengths $|A|$ and $|B|$ has an equivalent self-shrinking generator of length $L = 2 \cdot (|A| + |B|)$. Despite this similarity, the self-shrinking generator has shown even more resistance to cryptanalysis than the shrinking generator [18].

3.2. p-ary Generalized Self-Shrinking Generator. The proposed p-ary generalized Self-Shrinking Generator (pGSSG), given in Figure 4, consists of a pLFSR generator A, whose length will be denoted by L. It generates sequence $(a_i)_{i\geq 0}$ with p-ary digits (i.e., $(a_i)_{i\geq 0}$, $0 \leq a_i \leq p-1$) and $0 \leq i \leq L-1$. The multipliers of the feedbacks are given by coefficients $q_1, q_2, \ldots, q_L$, $q_L \in [0, 1, \ldots, p-1]$ of the primitive polynomial in $\mathrm{GF}(p^L)$. Every element can remember one p-ary number. The register is initialized by p-ary sequence $(a_0, a_1, \ldots, a_{L-1})$.

The pGSSG selects a portion of the output p-ary LFSR sequence by controlling the p-ary LFSR itself using the following algorithm.

Definition 4. The algorithm of the p-ary Generalized Self-Shrinking Generator (Figure 5) consists of the following steps.

(1) The p-ary LFSR is clocked with clock sequence with period T.

(2) The output pLFSR sequence is split into p-tuples $a_{pi}, a_{pi+1}, a_{pi+2}, \ldots, a_{pi+(p-1)}$, $i = 0, 1, \ldots$.

(3) If $a_{pi} = 0$ the whole p-tuple is discarded from the GSSG output, that is, the output is shrunken.

(4) When $a_{pi} \neq 0$, the corresponding digit $a_{pi+a_{pi}}$ in the p-tuple forms the output of the GSSG. For example, if $a_{pi} = 1$, then a_{pi+1} is output and the other digits a_{pi}, $a_{pi+2}, \ldots, a_{pi+(p-1)}$ are discard. If $a_{pi} = 2$, then a_{pi+2} is output and the other digits $a_{pi}, a_{pi+1}, a_{pi+3}, \ldots$, $a_{pi+(p-1)}$ are discard and so on. If $a_{pi} = p-1$, then $a_{pi+(p-1)}$ is output and the other digits $a_{pi}, a_{pi+1}, \ldots$, $a_{pi+(p-2)}$ are discard.

(5) The shrunken p-ary GSSG output sequence is transformed into binary sequence in which every p-ary number is presented with $\lceil \log_2(p-1) \rceil$ binary digits, where $\lceil x \rceil$ is the smallest integer which is greater or

Table 1: Binary transformation of p-ary digit.

p-ary number	Binary presentation of p-ary digit				
	$p = 3$	$p = 5$	$p = 7$	$p = 11$	$p = 13$
1	0	00	001	0011	0010
2	1	01	010	0100	0011
3	—	10	011	0101	0100
4	—	11	100	0110	0101
5	—	—	101	0111	0110
6	—	—	110	1000	0111
7	—	—	—	1001	1000
8	—	—	—	1010	1001
9	—	—	—	1011	1010
10	—	—	—	1100	1011
11	—	—	—	—	1100
12	—	—	—	—	1101

Table 2: Output of the 3GSSG.

3-tuple	Output (binary)	3-tuple	Output (binary)	3-tuple	Output (binary)
1**0**1	0 (0)	21**1**	1 (0)	20**1**	1 (0)
1**1**0	1 (0)	020		21**2**	2 (1)
21**0**	0 (1)	22**2**	2 (1)	001	
012		1**1**2	1 (0)	011	
1**0**0	0 (0)	20**2**	2 (1)	1**2**2	2 (1)
1**0**2	0 (1)	22**0**	0 (0)	010	
1**2**1	2 (1)	1**2**0	2 (1)	1**1**1	1 (0)
002		021		22**1**	1 (0)
022		20**0**	0 (1)		

equal to x. Every output number i from 1 to $p-1$ of p-ary GSSG sequence is depicted with p-ary expansion of the number:

$$(i-1)+\frac{2^{\lceil \log_2 p-1\rceil}-p-1}{2}. \tag{10}$$

For example the transformation of every p-ary digit for $p = 3, 5, 7, 11, 13$ is done according to the Table 1.

(6) Every p-ary zero in its i appearance ($i = 1, 2, 3, \ldots$) of the generated p-ary sequence can be represented binary by number d_i,

$$d_i = \begin{cases} (d_{i-1}+1) \bmod p & \text{if } d_{i-1} < p-1 \\ 1 & \text{if } d_{i-1} = p-1 \end{cases} \tag{11}$$

and initial condition $d_0 = 0$.

3.3. Examples

3.3.1. 3GSSG. Let us chose Extended Galois field GF(3^3) and the primitive polynomial that generates GF(3^3) is a^3+2a^2+1. Then the corresponding pLFSR (i.e., 3LFSR) will be as shown in Figure 6.

The feedback polynomial is $q(x) = 2x^3 + x^2 - 1$ and all algebraic function is done modulo p (i.e., $p = 3$).

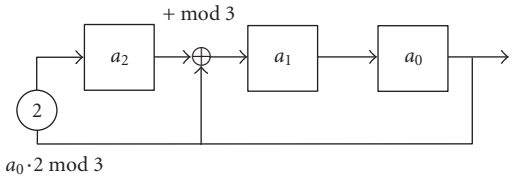

Figure 6: 3LFSR for GF(3^3) and primitive polynomial $a^3 + 2a^2 + 1$.

Let the initial state be $[a_2, a_1, a_0] = [0, 0, 1]$.

For all possible 3-tuples generated by the 3LFSR the 3GSSG output for a period will be [011120210220022110] and the way of its derivation according to the given algorithm is shown in Table 2.

The period of sequence is $T = 18$. The output is chosen from the value of the first symbol of 3-tuple, which defines which symbol forms the 3GSSG output. It is pointed out by the bold digits. Then the 3-ary output sequence is transformed into binary by the rules 5 and 6 in the pGSSG algorithm for $p = 3$ given in Table 1.

The binary output is given in the brackets in Table 2.

3.3.2. 5GSSG. Let us chose Extended Galois field GF(5^3) and primitive polynomial is $a^3 + 3a^2 + 2$. The corresponding pLFSR (i.e., 5LFSR) is shown in Figure 7.

Table 3: Output of the 5GSSG.

5-tuple	Output (binary)	5-tuple	Output (binary)	5-tuple	Output (binary)
1**0**222	0(00)	1**1**032	1(00)	3324**3**	4(11)
3142**1**	2(01)	00202		4230**2**	2(01)
1**2**044	2(01)	4220**1**	1(00)	4114**3**	3(10)
1**1**234	1(00)	20**0**40	0(11)	4341**0**	0(10)
4240**3**	3(10)	334**4**0	4(11)	23**2**23	2(01)
1**2**241	2(01)	340**0**3	0(00)	03132	
03430		1**1**133	1(00)	04144	
1**2**443	2(01)	21**3**00	3(10)	1**0**121	0(11)
4013**1**	1(00)	02221		1**0**323	0(00)
3243**3**	3(10)	1**4**210	4(11)	02024	
23**0**21	0(01)	20**4**44	4(11)	22**0**14	0(01)
1**1**431	1(00)	1**2**342	2(01)	00404	
3410**4**	0(10)	24**0**33	0(01)	3440**2**	0(10)

The feedback polynomial is $q(x) = 2x^3 + x^2 - 1$ and all algebraic function is done modulo 5 (i.e., $p = 5$).

Let the initial state be $[a_2, a_1, a_0] = [1, 1, 0]$.

For a part of all possible 5-tuples generated by the 5LFSR the output of the 5SSG generator for a part of period will be [01422213100342201230103404012000]. The way of its derivation according to the given algorithm is shown in Table 3.

The period is $T = 100$. The output is chosen from the value of the first symbol of 5-tuple, which defines which position symbol from this point to be selected to exit. Finally the 5-ary output sequence is transformed into binary by the rules 5 and 6 in the pGSSG algorithm for $p = 5$ given in Table 1. The binary output is given in the brackets in Table 3.

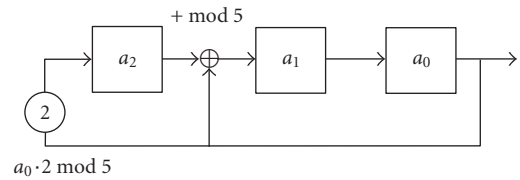

Figure 7: 3LFSR for GF(5^3) and primitive polynomial $a^3 + 3a^2 + 2$.

4. Properties of the pGSSG Output Sequence

The research of the p-ary GSSG has been modeled with Visual C++. To analyze the properties of the p-ary GSSG output sequence have been made more than 1000 tests with prime p up to 257 and various primitive feedback polynomials and initial states.

In this section first the pGSSG period is established. Then is proven that the output pGSSG sequence is balanced, that is, the number of 1s and 0s are equal in a period of pGSSG sequence. In Section 4.3 the minimum length of the used pLFSR registers to guarantee a key length of 512 and 1024 bits, which is useful for cryptographic applications nowadays, is calculated. Finally the good randomness properties of the pGSSG output sequences are proven by the results of the statistical NIST test suite.

4.1. Period of pGSSG. Hence a p-ary LFSR sequence is with period $T = p^L - 1$ then the pGSSG output sequence also is periodic. In fact after $p(p^L - 1)$ bits of the original sequence, the sequence of p-tuples $a_{pi}, a_{pi+1}, a_{pi+2}, \ldots, a_{pi+(p-1)}$ has been processed and the next p-tuple will be the same as first $a_0, a_1, a_2, \ldots, a_{p-1}$. As is well known, the p-tuples occurs balanced in one pm-sequence period. Therefore $1/p$ of all p-tuples will begin with bit $0, 1/p$ of all will begin with 1 and so on $1/p$ of all will begin with $p - 1$. Thus the p^{L-1} p-tuples will be discarding because they begin with 0 bit. Remaining $(p-1)p^{L-1}$ p-tuples will produce an one bit in the output sequence because this p-tuples begin with nonzero bits. Consequently the period of the p-ary GSSG output sequence will be

$$T_{p\text{GSSG}} = (p-1)p^{L-1} \tag{12}$$

p-ary digits.

When pGSSG output sequence is transformed into balanced binary sequence the period will be

$$T_{2\text{GSSG}} = T_{p\text{GSSG}} \left\lceil \log_2(p-1) \right\rceil \tag{13}$$

bits.

4.2. Number of Elements in p-Ary GSSG Sequence. Since the pm-sequence, produced by the p-ary LFSR is balanced and the pGSSG output sequence also is balanced the number of the each elements $0, 1, \ldots, p-1$ is equal to

$$N_i = (p-1)p^{L-2}, \quad i = 0, 1, \ldots, p-1. \tag{14}$$

After binary transformation the balanced property is in fact also, thus the number of zeros and ones are equal to

$$N_{b0} = N_{b1} = (p-1)p^{L-2} \cdot \left\lceil \log_2(p-1) \right\rceil. \tag{15}$$

The experimental results for different pGSSG and their parameters are given in Table 4.

Table 4: Results for simulated 3GSSG and 5GSSG.

GF(p^n)	Primitive polynomial	$T_{p\text{LFSR}}$	$T_{p\text{GSSG}}$	N_0	N_1	N_2	N_3	N_4	$T_{2\text{GSSG}}$	N_{b0}	N_{b1}
GF(3^3)	$a^3 + 2a^2 + 1$	26	18	6	6	6			18	9	9
GF(3^4)	$a^4 + 2a^3 + 2$	80	54	18	18	18			54	27	27
GF(3^5)	$a^5 + 2a^2 + a + 1$	242	162	54	54	54			162	81	81
GF(3^5)	$a^5 + 2a + 1$	242	162	54	54	54			162	81	81
GF(3^6)	$a^6 + a + 2$	728	486	162	162	162			486	243	243
GF(3^7)	$a^7 + a^2 + 2a + 1$	2186	1458	486	486	486			1458	729	729
GF(5^3)	$a^3 + 3a^2 + 2$	124	100	20	20	20	20	20	200	100	100
GF(5^4)	$a^4 + a^2 + 2a + 3$	624	500	100	100	100	100	100	1000	500	500
GF(5^5)	$a^5 + 4a + 2$	3124	2500	500	500	500	500	500	5000	2500	2500
GF(5^6)	$a^6 + a + 2$	15624	12500	2500	2500	2500	2500	2500	25000	12500	12500
GF(5^7)	$a^7 + 3a + 2$	78124	62500	12500	12500	12500	12500	12500	125000	62500	62500

4.3. Key for Encryption. The encryption key for the pGSSG is an initial state $(a_0, a_1, \ldots, a_{L-1})$ of the pLFSR register. Thus the encryption key will be long L p-ary digits.

One p-ary digit may be presented by $\lfloor \log_2 p \rfloor$ bits, where $\lfloor x \rfloor$ denote the smallest integer greater and equal to x. Consequently length of the encryption key is

$$L_{\text{Key}} = L \cdot \left\lfloor \log_2 p \right\rfloor. \tag{16}$$

As one can see the greater p is the longer the key is. Table 5 shows the minimum length of the used pLFSR register to guarantee a key length of 512 and 1024 bits, which are useful for cryptographic applications nowadays.

4.4. Statistical Properties. To test statistical properties of the keystreams generated by the pGSSG with prime p up to 257, 1000 sequences of length 10^6 have been tested by the National Institute of Standards and Technology (NIST) statistical test suite [23]. Table 6 shows the NIST test results for two sets of 100 keystreams each of length 10^6 for $p = 17$ and $p = 257$. To generate these 100 keystreams randomly chosen primitive polynomials and initial states are used.

The results demonstrate that pGSSG keystreams have good randomness properties, that is, they are well balanced, uniform, scalable and uncompressible.

4.5. Implementation. Unfortunately, the hardware implementation of the pGSSG is not as simple as the original SSG, because the algebraic operations are performed in the GF(p), $p \geq 3$. It takes $\lfloor \log_2 p \rfloor$ triggers to present one p-ary number. Hardware accelerators can perform the computationally intensive operations far quicker. Field-Programmable Gate Arrays (FPGAs) are well-suited for this application due to their reconfigurability and versatility.

The pGSSG is more suitable for software implementation. The cases with $p = 17$ and $p = 257$ are particularly suitable, because the binary output of the generator produces 4 or 8 bits at a time, respectively.

Table 5: Minimum length of pLFSR to guarantee a given min key length.

p	$L_{\min}$ for key length	
	512 bits	1024 bits
2	512	1024
3	256	512
5, 7	171	342
11, 13	128	256
17, 19, 23, 29, 31	103	205
37, 41, 43, 47, 53, 59, 61	86	171
67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127	74	147
131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251	64	128

5. Cryptanalysis

The goal of the stream cipher attack based on a clock controlled generators is to recover the secret key that includes the initial state of the used LFSRs. For better security the key may contain the feedback polynomial of the LFSRs. In this section, the security of the pGSSG against the exhaustive search and entropy attacks are analysed.

The suggested from Maier and Staffelbach two general attacks named exhaustive search attack and entropy attack [11] recover the initial state of the SSG from the knowledge of the short segment of the generated keystream requiring respectively $O(2^{0.79n})$ and $O(2^{0.75n})$ computational steps.

In these attacks we assume that the secret key consists only of the initial state of the p-ary GSSG.

5.1. Exhaustive Search Attack. Let $z_0, z_1, z_2, \ldots$ be a known portion of the binary keystream generated by pGSSG. As the prime p is also known the attacker can divide it by $\lceil \log_2 p - 1 \rceil$ bits. Each portion of $\lceil \log_2 p - 1 \rceil$ bits presents one p-ary number.

Let then $b_0, b_1, b_2, \ldots$ be a known p-ary portion of the pGSSG. The p-ary number b_j for some j is generated by

Table 6: Results from the NIST test suite for two sets of 100 keystreams of length 10^6, generated by the pGSSG.

NIST Statistical Test	17GSSG		257GSSG	
	P-value	Pass rate	P-value	Pass rate
Frequency	.798139	0.990000	.955835	1.000000
Block-frequency	.334538	0.990000	.834308	0.980000
Cumulative-sums 1	.637119	0.990000	.289667	1.000000
Cumulative-sums 2	.851383	0.990000	.595549	1.000000
Runs	.030806	1.000000	.867692	1.000000
Longest-run	.867692	0.970000	.162606	0.980000
Rank	.171867	1.000000	.437274	1.000000
FFT	.574903	1.000000	.191687	0.990000
Nonperiodic-templates*	.478512	0.988784	.504990	0.990676
Overlapping-templates	.096578	0.990000	.334538	0.990000
Universal	.213309	1.000000	.129620	1.000000
Apen	.000230	0.960000	.911413	1.000000
Random-excursions**	.246591	0.993663	.564158	0.993738
Random-excursions-variant***	.269248	0.992489	.479677	0.996289
Serial 1	.096578	1.000000	.437274	0.990000
Serial 2	.851383	1.000000	.574903	0.990000
Lempel-ziv	.012650	0.990000	.025193	0.980000
Linear-complexity	.739918	0.990000	.971699	0.970000

*The given values are average of the all 148 nonperiodic-templates NIST tests.
**The given values are average of the all 8 random-excursions NIST tests.
***The given values are average of the all 48 random-excursions-variant NIST tests.

the p-tuple $a_i, a_{i+1}, a_{i+2}, \ldots, a_{i+(p-1)}$ of the output pLFSR sequence $a_0, a_1, a_2, \ldots$, $a_i \in [0, 1, \ldots, p-1]$, where the index i is unknown.

From the knowledge of the first p-ary number b_0 one can conclude that for the first p-tuple there is $p-1$ possible cases:

$$\begin{aligned} &1, b_0, a_{i+2}, \ldots, a_{i+(p-1)}, \\ &2, a_{i+1}, b_0, \ldots, a_{i+(p-1)}, \\ &\qquad \ldots \\ &p-1, a_{i+1}, a_{i+2}, \ldots, b_0. \end{aligned} \tag{17}$$

For each case in (17) there are p^{p-2} possibilities for the $p-2$ unknown p-ary numbers. In other hand by definition 4 of the pGSSG algorithm for the p-ary number b_0 there are two possible cases: first, the one p-ary number from 1 to $p-1$, defined by (10) or second, the p-ary zero in its appearance by (11).

Therefore the number of possibilities for the first p-ary number b_0 is $2 \cdot (p-1) \cdot p^{p-2}$.

For the next known p-ary number b_1 there are the same $2 \cdot (p-1) \cdot p^{p-2}$ possible cases for the second p-tuple a_{p+i}, $a_{p+i+1}, a_{p+i+2}, \ldots, a_{p+i+(p-1)}$:

$$\begin{aligned} &1, b_1, a_{p+i+2}, \ldots, a_{p+i+(p-1)}, \\ &2, a_{p+i+1}, b_1, \ldots, a_{p+i+(p-1)}, \\ &\qquad \ldots \\ &p-1, a_{p+i+1}, a_{p+i+2}, \ldots, b_1. \end{aligned} \tag{18}$$

In addition to the above cases there are another p^{p-1} possibilities for which the output pGSSG sequence is shrunken

$$0, a_{p+i+1}, a_{p+i+2}, \ldots, a_{p+i+(p-1)}. \tag{19}$$

Hence all possible cases for the p-ary number b_1 are

$$V = 2 \cdot (p-1) \cdot p^{p-2} + p^{p-1}. \tag{20}$$

For each of these V possibilities there are another V possibilities for the next p-tuple and so on.

Therefore, for reconstructing m p-tuples, that is, $n = m \cdot p$ p-ary numbers, are needed

$$S = V^{m-1} \approx V^{n/p} \tag{21}$$

possible states for the pLFSR.

Example 1 ($p = 3$). The known bit portion of the sequence $z_0, z_1, z_2, \ldots$ is equal to the known p-ary portion $b_0, b_1, b_2, \ldots$ of the pGSSG because $\lceil \log_2(3-1) \rceil = 1$ bits. Every bit 0 can present the 3-ary bit 1 or the 3-ary zero. The bit 1 can present the 3-ary bit 2 or the 3-ary zero (see Table 1). From the known first bit b_0 there are two possible cases for the first triple a_i, a_{i+1}, a_{i+2}

$$\begin{aligned} &1, b_0, a_{i+2}, \\ &2, a_{i+1}, b_0. \end{aligned} \tag{22}$$

Table 7: A comparison of the complexity of the used attack for SSG, MSSG and pGSSG.

Attack	SSG	MSSG	pGSSG					
			$p = 3$	$p = 5$	$p = 7$	$p = 11$	$p = 13$	$p = 17$
Exhaustive search	$O(2^{0.79n})$	$O(2^{0.862n})$	$O(2^{1.464n})$	$O(2^{2.133n})$	$O(2^{2.612n})$	$O(2^{3.281n})$	$O(2^{3.532n})$	$O(2^{3.937n})$
Entropy	$O(2^{0.75n})$	$O(2^{0.833n})$	$O(2^{1.437n})$	$O(2^{2.066n})$	$O(2^{2.536n})$	$O(2^{3.210n})$	$O(2^{3.465n})$	$O(2^{3.879n})$

Each 3-ary number $a_i \in [0, 1, 2]$, that is, the six possible cases are

$$\begin{aligned} &1, b_0, 0 \qquad 2, 0, b_0 \\ &1, b_0, 1 \qquad 2, 1, b_0 \\ &1, b_0, 1 \qquad 2, 2, b_0 \end{aligned} \tag{23}$$

and another six possible cases in which b_0 is 3-ary zero are

$$\begin{aligned} &1, 0, 0 \qquad 2, 0, 0 \\ &1, 0, 1 \qquad 2, 1, 0 \\ &1, 0, 1 \qquad 2, 2, 0. \end{aligned} \tag{24}$$

For the next triple $a_{i+3}, a_{i+4}, a_{i+5}$ there are 12 possibilities of the same kind

$$\begin{aligned} &1, b_1, 0 \qquad 2, 0, b_1 \\ &1, b_1, 1 \qquad 2, 1, b_1 \\ &1, b_1, 1 \qquad 2, 2, b_1 \\ &1, 0, 0 \qquad 2, 0, 0 \\ &1, 0, 1 \qquad 2, 1, 0 \\ &1, 0, 1 \qquad 2, 2, 0 \end{aligned} \tag{25}$$

and 9 possible cases $0, a_{i+4}, a_{i+5}$ when the output is shrunken

$$\begin{aligned} &0, 0, 0 \qquad 0, 1, 0 \qquad 0, 2, 0 \\ &0, 0, 1 \qquad 0, 1, 1 \qquad 0, 2, 1 \\ &0, 0, 2 \qquad 0, 1, 2 \qquad 0, 2, 2. \end{aligned} \tag{26}$$

Therefore the possible cases are

$$V = 2 \cdot (p-1) \cdot p^{p-2} + p^{p-1} = 2.3.3 + 3^2 = 21, \tag{27}$$

which is listed above.

Then we calculate the possible pLFSR states

$$S \approx V^{n/p} \approx 21^{n/3} = 2^{(\log_2 21/3)n} = 2^{1.464n}. \tag{28}$$

Example 2 ($p = 5$). The number of possible cases for the 5-tuples are

$$V = 2 \cdot (p-1) \cdot p^{p-2} + p^{p-1} = 2.4.5^3 + 5^4 = 1625 \tag{29}$$

and states of the used pLFSR are

$$S \approx V^{n/p} \approx 1625^{n/5} = 2^{(\log_2 1625/5)n} = 2^{2.133n}. \tag{30}$$

5.2. Entropy Attack. The entropy of the p-tuples will be calculated. As mentioned above for the p-tuple $a_{p+i}, a_{p+i+1}, a_{p+i+2}, \ldots, a_{p+i+(p-1)}$ there are V possibilities, where $2 \cdot (p-1) \cdot p^{p-2}$ are in the form (18). In the half of these possibilities p-ary number $b_1 \in [1, 2, \ldots, p-1]$ and the probability of each of them is $(p-1)/p^p$. In the other half $b_1 = 0$ and each of them has probability $1/p^p$. The other p^{p-1} possibilities of the form (19) also have probability $1/p^p$.

Therefore, the entropy of the p-tuple is

$$\begin{aligned} H = &-(p-1) \cdot p^{p-2} \cdot \frac{(p-1)}{p^p} \log_2 \frac{p-1}{p^p} \\ &- (p-1) \cdot p^{p-2} \cdot \frac{1}{p^p} \log_2 \frac{1}{p^p} \\ &- p^{p-1} \cdot \frac{1}{p^p} \log_2 \frac{1}{p^p} \\ = &-\frac{(p-1)^2}{p^2} \cdot \log_2 \frac{p-1}{p^p} - \frac{p-1}{p^2} \cdot \log_2 \frac{1}{p^p} \\ &- \frac{1}{p} \cdot \log_2 \frac{1}{p^p} \\ = &-\frac{(p-1)^2}{p^2} \cdot \log_2 \frac{p-1}{p^p} \\ &- \left(\frac{p-1}{p^2} + \frac{1}{p} \right) \cdot \log_2 \frac{1}{p^p}. \end{aligned} \tag{31}$$

Because the entropy per one p-ary number is H/p, the entropy search among all different cases would require $2^{n \cdot H/p}$ steps.

For $p = 3$ the entropy of 3-tuples is

$$\begin{aligned} H_{p=3} &= -6. \frac{2}{27} \cdot \log_2 \frac{2}{27} - 15 \cdot \frac{1}{27} \log_2 \frac{1}{27} \\ &= 4.3104. \end{aligned} \tag{32}$$

Hence, the complexity of the entropy attack is $O(2^{n \cdot H/p}) = O(2^{1.43684n})$.

The complexity of exhaustive search and entropy attack of the pGSSG for primes up to 17 is compared to the SSG and recently proposed MSSG [8] in Table 7.

The results show that the pGSSG is more secure than SSG and MSSG against exhaustive search and entropy attack. The complexity of the used pGSSG attack increases with increasing the prime p.

6. Conclusions and Future Works

In this paper the generalization of the self-shrinking generator in Galois Field GF(p^n) for arbitrary prime p is proposed. The architecture of the new p-ary Generalized Self-Shrinking Generator is suggested. It is proved that the pGSSG has large period and good statistical properties. The experimental results analysis shows that the sequence generated by pGSSG is well balanced, uniform, scalable, uncompressible and unpredictable.

The complexity of the exhaustive search and entropy attack of the pGSSG is established. It is shown that the pGSSG is more secure than SSG and MSSG against exhaustive search and entropy attack. It is proven that the complexity of the used pGSSG attack increases with increasing the prime p.

Above-mentioned properties give the reason to consider the pGSSG as a pseudorandom generator that can be useful as a part of modern stream ciphers.

However, there are some theoretical and practical issues that need to be addressed. From a theoretical point of view, improved cryptanalysis of the pGSSG keystream sequences is necessary to be done and the complexity of other known self-shrinking attacks like attack using long keystream segment, BDD-based attack and guess-and-determine attack, must be investigated.

On the practical side, the hardware FPGA implementation of the pGSSG generator must be designed. It will provide faster execution of the algebraic operations in the GF(p) for any prime number $P \geq 3$.

Acknowledgment

The authors would like to thank the referees for their helpful comments.

References

[1] A. Afzal, *WiMAX Security (MAC and Physical Layer), NYC OEM Consultation Project*, Spring, Willits, Calif, USA, 2006.

[2] S. W. Golomb and G. Gong, *Signal Design for Good Correlation: For Wireless Communication, Cryptography, and Radar*, Cambridge University Press, New York, NY, USA, 2005.

[3] M. K. Chetry and W. B. Vasantha Kandaswamy, "A note on self-shrinking lagged fibonacci generator," *International Journal of Network Security*, vol. 11, no. 1, pp. 11–13, 2010.

[4] D. Coppersmith, H. Krawczyk, and Y. Mansour, "The shrinking generator," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '93)*, vol. 773, pp. 22–39, Springer, Berlin, Germany, 1993.

[5] E. J. Dudewicz and T. G. Ralley, *The Handbook of Random Number Generation and Testing with TESTRAND Computer Code*, American Sciences Press, Columbus, Ohio, USA, 1981.

[6] A. Fúster-Sabatera and P. Caballero-Gil, "Analysis of the generalized self-shrinking generator," *Computers & Mathematics with Applications*, vol. 61, no. 4, pp. 871–880, 2011.

[7] G. Guang, *Sequence Analysis, Department of Combinatorics and Optimization*, University of Waterloo, Ontario, Canada, 1999.

[8] A. Kanso, "Modified self-shrinking generator," *Computers and Electrical Engineering*, vol. 36, pp. 993–1001, 2010.

[9] H. Krawczyk, "The shrinking generator: some practical considerations," in *Proceedings of the International Workshop on Fast Software Encryption*, R. Andersen, Ed., vol. 809 of *Lecture Notes in Computer Science*, pp. 45–46, Springer, Berlin, Germany, 1994.

[10] J. L. Massey, "Some applications of coding theory in cryptography," in *Codes and Ciphers: Cryptography and Coding IV*, P. G. Farrell, Ed., pp. 33–47, Formara, Essex, UK, 1995.

[11] W. Meier and O. Staffelbach, "The self-shrinking generator," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '94)*, A. De Santis, Ed., vol. 950 of *Lecture Notes in Computer Science*, pp. 205–214, Springer, Berlin, Germany, 1995.

[12] B. Schneier, *Applied Cryptography*, John Wiley & Sons, New York, NY, USA, 1998.

[13] B. Stoyanov, A. Milev, and A. Nachev, "Research on the self-shrinking 2-adic cryptographic generator," *Journal of Communication and Computer*, vol. 7, no. 11, pp. 67–71, 2010.

[14] I. Shparlinski, "On some properties of the shrinking generator," *Designs, Codes, and Cryptography*, vol. 23, no. 2, pp. 147–155, 2001.

[15] S. R. Blackburn, "The linear complexity of the self-shrinking generator," *IEEE Transactions on Information Theory*, vol. 45, no. 6, pp. 2073–2077, 1999.

[16] B. Stoyanov, "Improved cryptanalysis of the self-shrinking p-adic cryptographic generator," in *Proceedings of the International Conference Information Research and Applications*, M. Krassimir, I. Krassimir, and M. Ilia, Eds., pp. 112–115, Varna, Bulgaria, July 2008.

[17] M. J. Mihaljevic, "A faster cryptanalysis of self-shrinking generator," in *Proceedings of the Australasian Conference on Information Security and Privacy (ACISP '96)*, J. Pieprzyk and J. Seberry, Eds., vol. 1172 of *Lecture Notes in Computer Science*, pp. 182–189, Springer, Berlin, Germany, 1996.

[18] E. Zenner, M. Krause, and S. Lucks, "Improved cryptanalysis of self-shrinking generator," in *Proceedings of the Australasian Conference on Information Security and Privacy (ACIPS ;01)*, vol. 2119 of *Lecture Notes in Computer Science*, pp. 21–35, Springer, Berlin, Germany, 2001.

[19] M. Krause, "BDD-based cryptanalysis of keystream generators," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '02)*, vol. 2332/2002 of *Lecture Notes in Computer Science*, pp. 222–237, Springer, Berlin, Germany, 2002.

[20] M. Hell and T. Johansson, "Two new attacks on the self-shrinking generator," *IEEE Transactions on Information Theory*, vol. 52, no. 8, pp. 3837–3843, 2006.

[21] B. Zhang and D. Feng, "New guess-and-determine attack on the self-shrinking generator," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT '06)*, vol. 4284/2006, pp. 54–68, Springer, Berlin, Germany, 2006.

[22] T. Tashev and B. Bedzhev, "Modeling of the linear feedback shift registers in the galois field with arbitrary prime base," in *Proceedings of the International Scientific and Applied Science Conference*, pp. 261–265, 2006.

[23] A. Rukhin, J. Soto, J. Nechvatal et al., Special Publication 800-22, Revision 1a, A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, April 2010, ps. 131, http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf.