

Optimizing the Placement of Tap Positions

Enes Pasalic¹, Samir Hodžić^{1(✉)}, Samed Bajrić¹, and Yongzhuang Wei²

¹ University of Primorska, FAMNIT and IAM, Koper, Slovenia
enes.pasalic6@gmail.com, samir.hodzic@famnit.upr.si, samed.bajric@upr.si

² Guilin University of Electronic Technology, Guilin, P.R. China,
and Also with the State Key Laboratory of Integrated Services Networks,
Xidian University, Xi'an 710071, People's Republic of China
walker_wei@msn.com

Abstract. Although there are many different approaches used in cryptanalysis of nonlinear filter generators, the selection of tap positions in connection to guess and determine cryptanalysis has not received enough attention yet. In a recent article [18], it was shown that the so-called filter state guessing attack (FSGA) introduced in [15], which applies to LFSR based schemes that use (vectorial) Boolean filtering functions, performs much better if the placement of tap positions is taken into account. In this article, for a given LFSR of length L , we analyze the problem of selecting n (where $n \ll L$) tap positions of the driving LFSR (used as binary inputs to a filtering function) optimally so that the complexity of FSGA like attacks is maximized. An algorithm which provides a suboptimal solution to this problem is developed and it can be used for real-life applications when the choice of tap positions is to be made.

Keywords: Stream ciphers · Filtering generator · Guess and determine cryptanalysis · Filter state guessing attack · Tap positions

1 Introduction

Nonlinear filter generator is a typical representative of a hardware oriented design in stream ciphers. It consists of a single linear feedback shift register (LFSR) and a nonlinear function $F : GF(2)^n \rightarrow GF(2)^m$ that processes a fixed subset of n stages of the LFSR. This fixed subset of the LFSR's cells is usually called the *taps*.

There are many cryptanalytic approaches that have been applied to nonlinear filter generators during the last two decades. These methods mainly use the cryptographic weaknesses of the filtering function giving rise to Berlekamp-Massey linear complexity attacks [10], linear distinguishing and inversion attacks of Golić [5–7], algebraic attacks [4], probabilistic algebraic attacks [2, 17], and so on. To protect a nonlinear filter generator against these attacks, the filtering function should satisfy multiple cryptographic criteria that include high nonlinearity, high algebraic degree [14], high algebraic immunity (AI) [11], and many others.

Apart from resisting the attacks using the properties of the filtering function, a nonlinear filter generator should also have sufficient security margins against other generic cryptanalytic methods, e.g. time-memory-data tradeoff attacks [1, 8, 9], and guess and determine attacks. A classical guess and determine attack is a method based on guessing some portion of the secret key (state bits) in order to decrease the complexity of obtaining the remaining unknown key (state) bits. Recently, a new guess and determine attack, named Filter State Guessing Attack (FSGA), was introduced in [15]. The basic idea behind the FSGA is to perform a guess and determine attack on the preimage space of the filtering function $F : GF(2)^n \rightarrow GF(2)^m$. Since for uniformly distributed F there are 2^{n-m} such preimages, for any observed m -bit output block the attacker may for each choice of 2^{n-m} many possible inputs (over the whole set of sampling instances) set up an overdefined system of linear equations in secret state bits. This attack turns out to be successful only for relatively large m , more precisely for approximately $m > n/2$.

In certain cases, the running time of the FSGA may be lower than the running time of a classical algebraic attack (cf. [15]). In particular, a superior performance of the FSGA over classical algebraic attacks was demonstrated in the case the filtering function belongs to a class of vectorial Maiorana-McFarland functions (see e.g. [3]). Notice that the tap positions of a nonlinear filter generator are of no importance for the FSGA in [15]. More precisely, only one bit of the information was considered to be known from the previous sampling points. The complexity of the attack was significantly improved in [18], where the information from the neighbouring taps, in the attack named GFSGA (Generalized FSGA), was used for a further reduction of the preimage space. In particular, the attack complexity of GFSGA is very sensitive to the tap placements, though no algorithm for their choice was provided in [18]. The reader should however notice that there exist other kind of attacks on nonlinear filtering generators such as e.g. decimation attacks [12] and attacks that take the advantage of the normality of Boolean functions [13] whose complexity does not depend on the choice of tap positions.

The main motivation for this work relies on the fact that even after more than two decades of extensive research on the security and design of filtering generators the selection of tap positions has not been rigorously treated yet. The designers, well aware of the fact that a proper tap selection plays an important role in the design, mainly use some standard (heuristic) design rationales such as taking the differences between the positions to be prime numbers (if possible), the taps are distributed over the whole LFSR etc. Intuitively, selecting the taps at some consecutive positions of the LFSR should be avoided, and similarly placing these taps at the positions used for the realization of the feedback connection polynomial is not a good idea either. Another common criterion is to ensure that a multiset of differences of the tap positions is mutually coprime. This means, that for a given set of tap positions $\mathcal{I} = \{i_1, i_2, \dots, i_n\}$ of an LFSR of length L (thus $1 \leq i_1 < i_2 < \dots < i_n \leq L$) all the elements in the difference set formed as $\mathcal{D} = \{i_j - i_l : i_j, i_l \in \mathcal{I}, i_j > i_l\}$ are mutually coprime. In many situations, in real-life applications, this condition turns out to be hard to satisfy. To the best of our knowledge, no algorithm for determining an optimal tap placement, for given n and L , has been provided so far.

In this article, we firstly demonstrate some potentially misleading design rationales from the security point of view and discuss the complexity issues related to optimality. Indeed, for a standard size of an LFSR used in these schemes, say $L = 256$, and a recommended number of inputs $n \geq 16$, any exhaustive search over the set of $\binom{L}{n}$ elements is clearly infeasible. Therefore, we propose a suboptimal algorithm for this purpose, which at least when applied to LFSRs of relatively short length performs optimally (giving the best choice over all possibilities). It is also shown that certain choices of tap positions in real-life stream ciphers such as SOBER-t32 and SFINX could have been (slightly) further optimized with respect to guess and determine cryptanalysis, in particular their resistance to GFSGA would have been better.

The rest of the article is organized as follows. In Sect. 2, basic definitions regarding Boolean functions and the mathematical formalism behind their use with LFSRs is given. A brief overview of FSGA and GFSGA is given in Sect. 3. Section 4 discusses the relation between the complexity of the GFSGA attack and the number of repeated equations used in the reduction of the preimage space. Two versions of the algorithm for determining (sub)optimal tap positions for a given n and L are presented in Sect. 5, and their application for the choice of tap positions in SOBER-t32 and SFINX is discussed.

2 Preliminaries

A Boolean function is a mapping from $GF(2)^n$ to $GF(2)$, where $GF(2)$ denotes the binary Galois field and $GF(2)^n$ is an n -dimensional vector space spanned over $GF(2)$. A function $f : GF(2)^n \rightarrow GF(2)$ is commonly represented using its associated algebraic normal form (ANF) as follows:

$$f(x_1, \dots, x_n) = \sum_{u \in GF(2)^n} \lambda_u (\prod_{i=1}^n x_i^{u_i}),$$

where $x_i \in GF(2)$, ($i = 1, \dots, n$), $\lambda_u \in GF(2)$, $u = (u_1, \dots, u_n) \in GF(2)^n$. A vectorial (multiple output) Boolean function $F(x)$ is a mapping from $GF(2)^n$ to $GF(2)^m$, with ($m \geq 1$), which can also be regarded as a collection of m Boolean functions, i.e., $F(x) = (f_1(x), \dots, f_m(x))$. Commonly, $F(x)$ is chosen to be uniformly distributed, that is, $\#\{x \in GF(2)^n | F(x) = z\} = 2^{n-m}$, for all $z \in GF(2)^m$. Moreover, for any $z = (z_1, \dots, z_m) \in GF(2)^m$, we denote the set of preimage values by $S_z = \{x \in GF(2)^n | F(x) = z\}$.

2.1 Nonlinear Filtering Generator

A filtering generator consists of a single LFSR of length L whose n fixed positions (taps) are used as the inputs to a filtering function $F : GF(2)^n \rightarrow GF(2)^m$ (also represented as $F(x) = (f_1(x), \dots, f_m(x))$), thus outputting $m \geq 1$ keystream bits at the time. A general description of a filter generator is as follows:

$$(z_1^t, \dots, z_m^t) = (f_1(\ell_n(s^t)), \dots, f_m(\ell_n(s^t))),$$

where $\mathbf{s}^t = (s_0^t, \dots, s_{L-1}^t)$ is the secret state of the LFSR at time t , the notation $\ell_n(\mathbf{s}^t)$ means that a subset of n bits of $\mathbf{s}^t = (s_0^t, \dots, s_{L-1}^t)$ (at fixed positions) is passed as the input to Boolean functions f_1, \dots, f_m , and z_1^t, \dots, z_m^t are the corresponding output keystream bits.

Due to linearity of its feedback connection polynomial, at any $t > 0$ we have $\ell_n(s_0^t, \dots, s_{L-1}^t) = (\psi_1^t(\mathbf{s}), \dots, \psi_n^t(\mathbf{s}))$, where the linear functions $\psi_i^t(\mathbf{s}) = \sum_{j=0}^{L-1} a_{i,j}^t s_j$, ($i = 1, \dots, n$), are unique linear combinations of the initial secret state bits $\mathbf{s}^0 = (s_0, \dots, s_{L-1})$, at time $t = 0$. The LFSR is updated by computing the update bit s_L (as a linear combination of s_0, \dots, s_{L-1} determined by the connection polynomial) and shifting its content to the left (while at the same time outputting the bit s_0), so that $\mathbf{s}^1 = (s_1, \dots, s_L)$. The binary coefficients $a_{i,j}^t$ above can therefore be efficiently computed from the connection polynomial of LFSR for all $t \geq 0$.

3 Overview of FSGA and GFSGA

For self-completeness and due to the close relation with subsequent sections, we briefly describe the main ideas behind FSGA and its extension GFSGA. For both attacks there is no restriction on $F : GF(2)^n \rightarrow GF(2)^m$, thus F satisfies all the relevant criteria including a uniform distribution of its preimages.

3.1 FSGA Description

For every observation of the cipher output $z^t = (z_1^t, \dots, z_m^t)$ at time t , there are 2^{n-m} possible inputs $x^t \in S_{z^t}$. Moreover, for every guessed preimage $x^t = (x_1^t, \dots, x_n^t) \in S_{z^t}$, one obtains n linear equations in the secret state bits s_0, \dots, s_{L-1} through $x_i^t = \sum_{j=0}^{L-1} a_{i,j}^t s_j$, for $1 \leq i \leq n$. The goal of the attacker is to recover the initial state bits (s_0, \dots, s_{L-1}) after obtaining sufficiently many keystream blocks $z^t = (z_1^t, \dots, z_m^t)$. If the attacker observes the outputs at the time instances t_1, \dots, t_c , so that $nc > L$, then with high probability each system of nc linear equations is independent but only one system will provide a consistent (correct) solution.

As there are $2^{(n-m)c}$ possibilities of choosing c input tuples $(x_1^{t_1}, \dots, x_n^{t_1}), \dots, (x_1^{t_c}, \dots, x_n^{t_c})$, and for each such c -tuple a system of nc linear equations in L variables is obtained. The complexity of solving a single overdefined system of linear equations with L variables is about L^3 operations. Thus, the complexity of the FSGA is about $2^{(n-m)c} L^3$ operations, where $c \approx \lceil \frac{L}{n} \rceil$.

3.2 GFSGA Description

The major difference to FSGA is that the GFSGA method efficiently utilizes the tap positions of the underlying LFSR. Let the tap positions of the LFSR be specified by the set $\mathcal{I}_0 = \{i_1, i_2, \dots, i_n\}$, $1 \leq i_1 \leq i_2 \leq \dots \leq i_n \leq L$. If at the time instance t_1 , we assume that the content of the LFSR at these tap

positions is given by $(s_{i_1}^{t_1}, \dots, s_{i_n}^{t_1}) = (a_1, \dots, a_n)$, then at $t = t_1 + \sigma$ we have $(s_{i_1+\sigma}^{t_1+\sigma}, \dots, s_{i_n+\sigma}^{t_1+\sigma}) = (a_1, \dots, a_n)$, where cutting modulo L can be performed if necessary. Notice that the state bits at positions $i_1 + \sigma, \dots, i_n + \sigma$ does not necessarily intersect with \mathcal{I}_0 , thus if the intersection is an empty set no information from the previous sampling can be used at the sampling instance $t_1 + \sigma$. However, we can always select σ so that at least one bit of information is conveyed. More formally, the observed outputs at t_1, \dots, t_c , where $t_i = t_1 + (i - 1)\sigma$ and $1 \leq \sigma \leq (i_n - i_1)$, may give rise to identical linear equations since the equations $x_i^{t_u} = \sum_{j=0}^{L-1} a_{i,j}^{t_u} s_j$ (where $1 \leq i \leq n$) may be shifted to $x_l^{t_v} = \sum_{j=0}^{L-1} a_{i,j}^{t_v} s_j$, for some $1 \leq i < l \leq n, 1 \leq u < v \leq c$.

It is of importance to determine how many identical linear equations will be obtained for all the sampling instances t_1, \dots, t_c . By introducing $k = \lfloor \frac{i_n - i_1}{\sigma} \rfloor$, and for $\mathcal{I}_0 = \{i_1, i_2, \dots, i_n\}$ defining recursively:

$$\begin{aligned} \mathcal{I}_1 &= \mathcal{I}_0 \cap \{i_1 + \sigma, i_2 + \sigma, \dots, i_n + \sigma\}, \\ \mathcal{I}_2 &= \mathcal{I}_1 \cup \{\mathcal{I}_0 \cap \{i_1 + 2\sigma, i_2 + 2\sigma, \dots, i_n + 2\sigma\}\}, \\ &\vdots \\ \mathcal{I}_k &= \mathcal{I}_{k-1} \cup \{\mathcal{I}_0 \cap \{i_1 + k\sigma, i_2 + k\sigma, \dots, i_n + k\sigma\}\}. \end{aligned} \tag{1}$$

the analysis in [18] showed that the complexity of the GFSGA is closely related to the parameter $r_i = \#\mathcal{I}_i$, where $i = 1, \dots, k$.

Remark 1. For instance, the above notation means that for some $i \in \mathcal{I}_1$ (and therefore $i \in \mathcal{I}_0$) the state bit $s_i^{t_2}$ was used in the previous sampling since it was at the position $i - \sigma \in \mathcal{I}_0$ at time t_1 , where $t_2 = t_1 + \sigma$. The idea is easily generalized for $\#\mathcal{I}_i = r_i$, where $i = 2, \dots, k$.

The number of identical equations obtained in [18] is given as follows. If $c \leq k$, then in total $\sum_{i=1}^{c-1} r_i$ identical linear equations are obtained, whereas for $c > k$ this number is $\sum_{i=1}^k r_i + (c - k - 1)r_k$. Note that in this case $r_k = r_{k+1} = \dots = r_{c-1}$ due to the definition of k , which simply guarantees that after k sampling instances the maximum (and constant) number of repeated equations is attained. Consequently, the time complexity of the attack for $c \leq k$ was estimated as,

$$\begin{aligned} T_{Comp.}^{c \leq k} &= 2^{(n-m)} \times 2^{(n-m-r_1)} \times \dots \times 2^{(n-m-r_{(c-1)})} \times L^3 \\ &= 2^{(n-m)c - \sum_{i=1}^{c-1} r_i} \times L^3, \end{aligned} \tag{2}$$

and similarly, if $c > k$, the time complexity for $c > k$ was given by

$$\begin{aligned} T_{Comp.}^{c > k} &= 2^{(n-m)} \times 2^{(n-m-r_1)} \times \dots \times \\ &\quad \times 2^{(n-m-r_k)} \times 2^{(n-m-r_k) \times (c-k-1)} \times L^3 \\ &= 2^{(n-m)c - (\sum_{i=1}^k r_i + (c-k-1)r_k)} \times L^3. \end{aligned} \tag{3}$$

Remark 2. If $n - m - r_i \leq 0$, for some $i \in \{1, \dots, k\}$, then the knowledge of these r_i bits allows the attacker to uniquely identify the exact preimage value

form the set of 2^{n-m} possible preimages, i.e., we assume $2^{(n-m-r_i)} = 1$ when $n - m - r_i \leq 0$.

Table 1 (cf. [18]) gives a complexity comparison of FSGA, GFSGA and CAA (Classical algebraic Attack). The tap positions and the sampling difference σ are given below:

- (1) $\{3, 8, 13, 16, 21, 29, 32, 37, 44, 52, 67, 79, 92, 106, 111, 125, 155\}$, $\sigma = 5, c = 23$.
- (2) $\{2, 7, 17, 25, 27, 31, 48, 58, 61, 73, 82, 91, 103, 115, 123, 134, 146, 156\}$, $\sigma = 3, c = 20$.

Table 1. Complexity comparison for different (n, m) and $(K = 80, L = 160)$.

(n, m)	$(17, 6)$	$(18, 7)$
FSGA	2^{123}	2^{121}
CAA	2^{75}	2^{75}
GFSGA	$2^{53.97}$	$2^{64.97}$

4 Complexity Versus the Number of Repeated Equations

The complexity of GFSGA, which is a generic attack for this particular encryption scheme, strongly depends on the choice of tap positions, see also [18]. Therefore, our goal is to maximize this complexity which is certainly related to the minimization of the parameters $r_i = \#\mathcal{I}_i$, but not completely equivalent. Notice that by optimizing the resistance of these schemes to GFSGA does not necessarily imply the optimality of tap selections, though for the targeted filtering generator we cannot see other reasonable approaches in the context of the guess and determine cryptanalysis.

Let R denotes the number of repeated equations regardless of this number being $\sum_{i=1}^{c-1} r_i$ for $c \leq k$, or $\sum_{i=1}^k r_i + (c - k - 1)r_k$ for $c > k$. From [18], it somehow appears that an (sub)optimal choice of tap positions is the one that minimizes the number of repeated equations R , which is a bit misleading as illustrated by the following example.

Example 1. Let the tap positions be given by $\mathcal{I}_0 = \{1, 5, 13, 25, 41, 65, 77\}$, for $L = 80$, $n = 7$, and $m = 3$. Computing the complexity $T_{Comp.}$ for all sampling differences $\sigma = 1, 2, 3, \dots, 76$, one can verify that the best choice of σ for the attacker is $\sigma = 12$, with the minimal complexity $T_{Comp.} \approx 2^{23.97}$ and having $R = 177$ as the number of repeated equations. However, the computation below shows that for $\sigma = 4$, $R = 353$ is maximum possible, but in that case $T_{Comp.} \approx 2^{27.97}$.

To see why $\sigma = 4$ is not optimal for the attacker, we first compute $r_i = \#\mathcal{I}_i$,

$$\begin{aligned} \mathcal{I}_1 &= \{5\}, \mathcal{I}_2 = \{5, 13\}, \mathcal{I}_3 = \{5, 13, 25, 77\}, \mathcal{I}_4 = \{5, 13, 25, 41, 77\}, \\ \mathcal{I}_5 &= \{5, 13, 25, 41, 77\}, \mathcal{I}_6 = \{5, 13, 25, 41, 65, 77\}, \\ \mathcal{I}_j &= \{5, 13, 25, 41, 65, 77\}, \text{ for } j = 7, 8, \dots, 61. \end{aligned}$$

The number of sampling points c , for $k = \lfloor \frac{77-1}{4} \rfloor = 19$, is determined from the condition $nc - (\sum_{i=1}^k r_i + (c - k - 1)r_k) > L$, i.e., $c = 62$ is the smallest positive integer satisfying the condition. The terms $2^{(n-m-r_i)} \neq 1$ in (3), for which $r_i < n - m$ so that the number of preimages is greater than one, only appear for $r_1 = 1$ and $r_2 = 2$, i.e.,

$$T_{Comp.} = 2^{(n-m)} \times 2^{(n-m-r_1)} \times 2^{(n-m-r_2)} \times L^3 \approx 2^{27.97}.$$

For $j = 3, \dots, 61$, we have $2^{(n-m-r_j)} = 1$, in accordance to Remark 2.

Similarly, for $\sigma = 12$, which implies that $k = 6$, we obtain $c = 37$ (where c is derived from $nc - (\sum_{i=1}^k r_i + (c - k - 1)r_k) > L$) and “only” $R = 177$ repeated equations. The intersection sets in this case are given as,

$$\begin{aligned} \mathcal{I}_1 &= \{13, 25, 77\}, \mathcal{I}_2 = \{13, 25, 65, 77\}, \mathcal{I}_3 = \{13, 25, 41, 65, 77\}, \\ \mathcal{I}_j &= \{13, 25, 41, 65, 77\}, \text{ for } j = 4, 5, \dots, 36. \end{aligned}$$

The complexity computation in this case involves only $r_1 = 3$, i.e.,

$$T_{Comp.} = 2^{(n-m)} \times 2^{(n-m-r_1)} \times L^3 \approx 2^{23.97}.$$

Notice that for $j = 2, \dots, 36$, we have $2^{(n-m-r_j)} = 1$.

Remark 3. A lower complexity in the above example (for a larger number of repeated equations) is entirely due to a low difference between n and m so that many of the repeated equations could not be efficiently used since the preimages could be identified uniquely even without using these equations.

More formally, if σ' gives the maximal possible value of R though the attack complexity is not minimal, and σ'' gives the minimal attack complexity without maximizing R , then it holds

$$\sum_{r_j \in H_{\sigma''}} (n - m - r_j) < \sum_{r_i \in H_{\sigma'}} (n - m - r_i) \quad (4)$$

where $H_{\sigma'} = \{r_i < n - m : r_i \text{ obtained by } \sigma', i = 1, 2, \dots, c - 1\}$ and $H_{\sigma''} = \{r_j < n - m : r_j \text{ obtained by } \sigma'', j = 1, 2, \dots, c - 1\}$. In the above example, we have $H_{\sigma'} = \{r_1, r_2\} = \{1, 2\}$ with $\sigma' = 4$, and $H_{\sigma''} = \{r_1\} = \{3\}$ with $\sigma'' = 12$, for which (4) holds.

Another problem related to the approach of finding the intersection sets given by (1) is that the information contained in R and the cardinalities r_i alone does not fully specifies the properties of the repeated equations. The equations corresponding to the numbers in the sets \mathcal{I}_i may be repeated and found in other sets \mathcal{I}_j , where $i \neq j$, and even though they efficiently reduce the preimage space they do not contribute to the rank of the systems of linear equations that need to be solved. An alternative method of tracking the repeated equations, illustrated in the example bellow, turns out to give a deeper insight to the problem of selecting the tap positions optimally.

Example 2. Let the tap positions be given by $\mathcal{I}_0 = \{l_1, l_2, l_3, l_4, l_5\} = \{1, 4, 8, 9, 11\}$, $L = 15$, and the sampling distance $\sigma = 2$. Let $\mathbf{s}^{t_i} = (s_{0+(i-1)\sigma}, s_{1+(i-1)\sigma}, \dots, s_{14+(i-1)\sigma})$, denote the LFSR state over $c = 10$ sampling instances $t_i = (i-1)\sigma$, for $i = 1, 2, \dots, 10$. Moreover, at these different sampling instances, we represent the output bits of LFSR s_0, s_1, \dots via their indices in \mathbb{N} , i.e., $s_k \rightarrow (k+1) \in \mathbb{N}$. For instance, in Table 2 the number 27 corresponds to the bit s_{26} which becomes a part of the LFSR state \mathbf{s}^{t_9} at position l_5 . The LFSR state bits at tap positions $\mathcal{I}_0 = \{l_1, l_2, l_3, l_4, l_5\}$ are illustrated in Table 2.

Table 2. The LFSR state bits at given tap positions for $\sigma = 2$.

States	l_1	l_2	l_3	l_4	l_5
\mathbf{s}^{t_1}	$s_0 \rightarrow 1$	$s_3 \rightarrow 4$	$s_7 \rightarrow 8$	$s_8 \rightarrow 9$	$s_{10} \rightarrow 11$
\mathbf{s}^{t_2}	$s_2 \rightarrow 3$	$s_5 \rightarrow 6$	$s_9 \rightarrow 10$	$s_{10} \rightarrow 11$	$s_{12} \rightarrow 13$
\mathbf{s}^{t_3}	$s_4 \rightarrow 5$	$s_7 \rightarrow 8$	$s_{11} \rightarrow 12$	$s_{12} \rightarrow 13$	$s_{14} \rightarrow 15$
\mathbf{s}^{t_4}	$s_6 \rightarrow 7$	$s_9 \rightarrow 10$	$s_{13} \rightarrow 14$	$s_{14} \rightarrow 15$	$s_{16} \rightarrow 17$
\mathbf{s}^{t_5}	$s_8 \rightarrow 9$	$s_{11} \rightarrow 12$	$s_{15} \rightarrow 16$	$s_{16} \rightarrow 17$	$s_{18} \rightarrow 19$
\mathbf{s}^{t_6}	$s_{10} \rightarrow 11$	$s_{13} \rightarrow 14$	$s_{17} \rightarrow 18$	$s_{18} \rightarrow 19$	$s_{20} \rightarrow 21$
\mathbf{s}^{t_7}	$s_{12} \rightarrow 13$	$s_{15} \rightarrow 16$	$s_{19} \rightarrow 20$	$s_{20} \rightarrow 21$	$s_{22} \rightarrow 23$
\mathbf{s}^{t_8}	$s_{14} \rightarrow 15$	$s_{17} \rightarrow 18$	$s_{21} \rightarrow 22$	$s_{22} \rightarrow 23$	$s_{24} \rightarrow 25$
\mathbf{s}^{t_9}	$s_{16} \rightarrow 17$	$s_{19} \rightarrow 20$	$s_{23} \rightarrow 24$	$s_{24} \rightarrow 25$	$s_{26} \rightarrow 27$
$\mathbf{s}^{t_{10}}$	$s_{18} \rightarrow 19$	$s_{21} \rightarrow 22$	$s_{25} \rightarrow 26$	$s_{26} \rightarrow 27$	$s_{28} \rightarrow 29$

Our goal is to determine when some equation (state bit) is repeated on the tap positions l_1, \dots, l_4 at the sampling instances t_i . Hence, we observe the repetition of all consecutive tap positions $l_{j+1} - l_j$, then the differences $l_{j+2} - l_j$, etc. Let D be a set of all differences between consecutive tap positions, i.e.,

$$D = \{d_j | d_j = l_{j+1} - l_j, j = 1, 2, 3, 4\} = \{3, 4, 1, 2\}.$$

To consider all possible repetitions of the equations on all tap positions, we design a scheme of all possible differences:

Table 3. The scheme of all possible differences for the set D .

Row \ Columns	Col. 1	Col. 2	Col. 3	Col. 4
Row 1	d_1	d_2	d_3	d_4
Row 2	$d_1 + d_2$	$d_2 + d_3$	$d_3 + d_4$	
Row 3	$d_1 + d_2 + d_3$	$d_2 + d_3 + d_4$		
Row 4	$d_1 + d_2 + d_3 + d_4$			

Table 4. The scheme of all differences for $D = \{3, 4, 1, 2\}$.

Row\Columns	Col. 1	Col. 2	Col. 3	Col. 4
Row 1	3	4	1	2
Row 2	7	5	3	
Row 3	8	7		
Row 4	10			

In Table 3, Column 1 specifies the repetition of some equations at the tap position l_1 , Column 2 gives the repetition of equations on l_2 , etc. Similarly, Row 1 takes into account the consecutive repetitions from l_{i+1} to l_i , Row 2 regards the repetition from l_{i+2} to l_i , etc. In our example, by Table 3, we have Assuming the attacker starts the sampling with some step σ , the total number of repeated equations R is the sum of all equations which repeat on each of the tap positions l_j , where $j = 1, 2, 3, 4$.

Since Table 3 can be designed for an arbitrary set D , $\#D = n - 1$, the repetition of the same equations can be tracked as follows. We are looking for the first number in each column such that it is divisible by σ , which implies that we have the repetition of equations, otherwise there are no repetitions. Notice that in Table 4, in Column 1, $\sigma \nmid 3$, which implies that there is no repetition of equations from l_2 at l_1 . Also, since $2 \nmid 7$, there is no repetition from l_3 at l_1 . However, $2 \mid 8$, which implies that the equation(s) from l_4 will appear on l_1 after $\frac{8}{2} = 4$ sampling instances (cf. Table 2 where 9 appears at l_1 when the content of the LFSR is \mathbf{s}^{t_5}). Thereafter, one equation from l_4 appears at l_1 for every state \mathbf{s}^{t_i} , for $i \geq 5$. Further, the fact that $2 \mid 8$ and $2 \mid 10$ implies that $2 \mid d_4 = 2$, which means that we have a repetition from l_5 to l_1 at every LFSR state \mathbf{s}_i^t , $i \geq 2$. Since Column 1 already contains this number 8 which is divisible by 2, all the repeated equations from l_5 to l_1 are already taken into account, and we do not use number 10 (Table 4, Row 4) when calculating the number of repeated equations. So, $\frac{d_4}{2}$ is related to the repetitions of equations from l_5 to l_4 . Hence, the number of repeated equations R , for $c = 10$, is calculated as follows.

1. On l_1 , there are $(c - \frac{d_1+d_2+d_3}{\sigma}) = 10 - \frac{8}{2} = 6$ repeated equations.
2. On l_2 , there are $(c - \frac{d_2}{\sigma}) = 10 - 2 = 8$ repeated equations.
3. On l_3 , there are NO repeated equations, since we do not have the differences divisible by $\sigma = 2$.
4. On l_4 , there are $(c - \frac{d_4}{\sigma}) = 9$ repeated equations.

In total, we have $R = 6 + 8 + 0 + 9 = 23$ repeated equations.

The analysis performed in the above example leads to the following result concerning the number of repeated equations.

Proposition 1. Let $\mathcal{I}_0 = \{l_1, l_2, \dots, l_n\}$ be a set of tap positions, and let

$$D = \{l_{i+1} - l_i | i = 1, 2, \dots, n - 1\} = \{d_1, d_2, \dots, d_{n-1}\}.$$

The number of repeated equations is calculated as

$$R = \sum_{i=1}^{n-1} \left(c - \frac{1}{\sigma} \sum_{k=i}^m d_k \right), \quad (5)$$

where $\sigma \mid \sum_{k=i}^m d_k$ for some $m \in \mathbb{N}$, $i \leq m \leq n-1$ and $\frac{1}{\sigma} \sum_{k=i}^m d_k \leq c-1$. Moreover, if $\frac{1}{\sigma} \sum_{k=i}^m d_k \geq c$, for some $1 \leq i \leq n-1$, then $(c - \frac{1}{\sigma} \sum_{k=i}^m d_k) = 0$. This means that the repetition of the same equations (bits) starts to appear after the LFSR state s^{tc} .

Remark 4. The importance of the above proposition lies in a fact that the counting method of repeated equations does not depend on the relation between the number of sampling points c and k (where $k = \lfloor \frac{i_n - i_1}{\sigma} \rfloor$), i.e., it holds for both $c \leq k$ and $c > k$.

Notice that, in order to minimize the number of repeated equations, the terms $(c - \frac{1}{\sigma} \sum_{k=i}^m d_k)$, $i \leq m \leq n-1$, should be minimized. Hence, we want to avoid the divisibility by σ in the scheme of differences as much as possible. Moreover, for a given length L of LFSR, the differences between $d_i \in D$ should be maximized under the constraint $\sum_{i=1}^{n-1} d_i \leq L-1$, which is also conditioned by $1 \leq l_1 < l_2 < \dots < l_n \leq L$. In other words, the goal is to distribute the tap positions over entire LFSR while at the same time keeping the divisibility by σ as low as possible. Clearly, if $\sum_{i=1}^{n-1} d_i = L-1$, then $l_1 = 1$ and $l_n = L$.

5 Two Algorithms Towards an Optimal Selection of Taps

It turns out that the problem of optimizing the choice of \mathcal{I}_0 is closely related to the divisibility of the elements in the corresponding (multi)set of differences D by an arbitrary σ . Thus, instead of searching the set \mathcal{I}_0 directly, we focus on the set of differences D . The construction of the set D is however out of reach to be done exhaustively for moderately large L and n , and consequently we use some heuristic techniques to specify D (sub)optimally.

In what follows, we present a method of constructing the set D which gives a low number of repeated equations (confirmed by computer simulations) for every σ . The set D is specified using some heuristic design rationales (see below) and at the same time the differences d_i are maximized.

Step A: Find the elements of the set D . To do this and avoid the divisibility by σ , the following pattern is applied.

1. Prime numbers are the most favourable to join the set D . Since higher values of n dictate the repetitions of some elements in D , the repetition should be kept on minimum with a general tendency to choose co-prime differences. If some even numbers are taken, then the set D should contain just few of them, because they can result in many common (high) factors in the rows of Table 3.

2. Maximize the differences d_i under the constraint $\sum_{d_i \in D} d_i \leq L - 1$.

Step B: Find the best ordering of the chosen differences, which basically means that ordering of D is also important. This can be done using the following algorithm with the complexity $O(n! \cdot K)$, where K corresponds to the complexity of calculation $T_{Comp.}$ for all possible σ .

INPUT: The set D and the numbers L , $n = \#D + 1$ and m .

OUTPUT: The best ordering of the chosen differences, that is, an ordered set D that maximizes the complexity of the attack.

STEP 1: Generate a list of all permutations of the elements in D ;

STEP 2: For every permutation, find the minimal complexity for all steps σ from 1 to L ;

STEP 3: Generate a list of all minimal complexities from Step 2;

STEP 4: Find the maximal value in the list of all minimal complexities;

STEP 5: Return the corresponding permutation of the maximal value.

Open Problem 1. *Find an efficient algorithm, which returns the best ordering of the set D without searching all permutations.*

Remark 5. To measure the quality of a chosen set of differences D with respect to the maximization of $T_{Comp.}$ over all σ , the computer simulations indicate that an optimal ordering of the set D implies a small value of an optimal sampling distance σ . This is also a criterion that a set D is most likely chosen well (a sub-optimal choice). The term “most likely” concerns the difficulties of capturing the whole process of choosing the tap positions explicitly, due to a very complicated relation between σ , R , D and $T_{Comp.}$ through the scheme of differences. When choosing an output permutation (cf. Step 5 below), we always consider both σ and $T_{Comp.}$ though σ turns out to be a more stable indicator of the quality of a chosen set D .

Note that, the above algorithm performs an exhaustive search over all permutations of the input set. For practical values of L , usually taken to be $L = 256$, the time complexity of the above algorithm becomes practically infeasible already for $n > 10$. To reduce its factorial time complexity, we modify the above algorithm to process the subsets of the multiset D separately within the feasibility constraints imposed on the cardinalities of these subsets.

STEP 1: Choose a set X by **Step A**, where $\#X < \#D$ for which **Step B** is feasible;

STEP 2: Find the best ordering of X using the algorithm in **Step B** for $L_X = 1 + \sum_{x_i \in X} x_i < L$ and $m_X = \lfloor \#X \cdot \frac{m}{n-1} \rfloor$;

STEP 3: Choose a set Y by **Step A**, where $\#Y < \#D$ for which **Step B** is feasible;

STEP 4: “Generate” a list of all permutations of the elements in Y ;

- STEP 5:** Find a permutation (Y_p) from the above list such that for a fixed set X , the new set $Y_p X$ obtained by joining X to Y_p , denoted by $Y_p X$ (with the parameters $L_{Y_p X} = 1 + \sum_{x_i \in X} x_i + \sum_{y_i \in Y_p} y_i \leq L$ and $m_{Y_p X} = \lfloor \#Y_p X \cdot \frac{m}{n-1} \rfloor$), allows a small optimal step σ , in the sense of Remark 5;
- STEP 6:** If such a permutation, resulting in a small value of σ , does not exist in Step 5, then back to Step 3 and choose another set Y ;
- STEP 7:** Update the set $X \leftarrow Y_p X$, and repeat the steps 3 - 5 by adjoining new sets Y_p until $\#Y_p X = n - 1$;
- STEP 8:** Return the set $D = Y_p X$.

Remark 6. The parameters L_X and m_X are derived by computer simulations, where L_X essentially constrains the set X and m_X keeps the proportionality between the numbers m , $\#X$ and $\#D = n - 1$.

An illustration of our modified version of the above algorithm is given in the following example. Namely, for a rather practical choice of the parameters L , n and m , the whole procedure of defining the set of differences that eventually yields the tap positions is discussed. Some suboptimal choices of tap positions for varying input parameters L, n, m along with the time complexity of the GFSGA and the time complexity of applying our algorithms are given in Appendix (cf. Tables 5 and 6).

Example 3. Let $n = 17$, $m = 6$, and $F(x) : GF(2)^{17} \longrightarrow GF(2)^6$. Let $L = 160$ bits, the length of the secret key is $K = 80$ bits.

Let $X = \{5, 13, 7, 26, 11, 17\}$ be obtained using the algorithm in **Step B** for $L_X = 80$, $m_X = 2$. Let $Y = \{1, 2, 9, 15, 23\}$. Then, a permutation $Y_p = \{9, 1, 2, 23, 15\}$, i.e., the set

$$Y_p X = \{9, 1, 2, 23, 15, 5, 13, 7, 26, 11, 17\},$$

where $L_{Y_p X} = 130$ and $m_{Y_p X} = 4$, gives that $\sigma = 1$ is an optimal sampling distance for the attacker. Since $L_{Y_p X} \leq 160$, then we choose the set $Z = \{3, 4, 5, 7, 11\}$. Then, a permutation $Z_p = \{5, 11, 4, 3, 7\}$, i.e., the set $Z_p Y_p X = \{5, 11, 4, 3, 7, 9, 1, 2, 23, 15, 5, 13, 7, 26, 11, 17\}$, where $L_{Z_p Y_p X} = L = 160$ and $m_{L_{Z_p Y_p X}} = m = 6$ gives the optimal step $\sigma = 1$ for the attacker. Then we have

$$D = \{5, 11, 4, 3, 7, 9, 1, 2, 23, 15, 5, 13, 7, 26, 11, 17\},$$

and thus

$$\mathcal{I}_0 = \{1, 6, 17, 21, 24, 31, 40, 41, 43, 66, 81, 86, 99, 106, 132, 143, 160\}.$$

Hence, $\sigma = 1$ is optimal, with the minimal complexity $T_{Comp.} = 2^{86.97}$, which is essentially an extremely good choice of tap positions (non-exhaustively confirmed to be an optimal choice).

In what follows, we apply the above algorithms to two well-known stream cipher SOBER-t32 [16], [18] and SFINX [19].

SOBER-t32: An application of the GFSGA attack on unstuttered SOBER-t32 was considered in [18]. The tap positions of SOBER-t32 are given by $\mathcal{I}_0 = \{1, 4, 11, 16, 17\}$ (corresponding to the reverse order of the taps $1 \leftarrow s_{16}, 4 \leftarrow s_{13}$, etc.) and the sampling distance used in [18] was $\sigma = 3$. Due to the reverse order of the bits s_i , we consider the set D in reverse order, i.e. $D = \{1, 5, 7, 3\}$ instead of $\{3, 7, 5, 1\}$, since this ordering corresponds to our consideration of the LFSR states presented in Table 2. Regarding the set D , the set of all $r_i = \#\mathcal{I}_0$ is $\{1, 1, 2, 2, 3, 3, 4, 4, 4, 4, \dots\}$, i.e. $r_1 = r_2 = 1, r_3 = r_4 = 2, r_5 = r_6 = 3$ and $r_k = 4, k \geq 7$. At each sampling point we derive $40 - 8 \times r_i$ linear equations (cf. [18]). Therefore, the number of repeated equations is given by

$$40 + 32 + 32 + 24 + 24 + 16 + 16 + 8 \times (c - 7) + c, \quad (6)$$

which for $c = 47$ gives $R = 550$ linear equations (6). Thus the complexity of the attack can be estimated as

$$T_D = (17 \times 32)^3 \times 2^{35} \times 2^{2 \times 27} \times 2^{2 \times 19} \times 2^{2 \times 11} \times 2^{39 \times 3} = (17 \times 32)^3 \times 2^{266}.$$

Since $\#D = 4$, we can easily apply **Step A** and **Step B**, to come up with the new set $D^* = \{5, 2, 7, 2\}$, and get the set $\{0, 2, 2, 2, 3, 3, 4, 4, 4, 4, \dots\}$ of all $r_i = \#\mathcal{I}_0$. The inequality

$$40 + 40 + 32 + 32 + 32 + 24 + 24 + 8 \times (c - 7) + c \geq 544$$

implies $c = 42$, and $R = 546$ equations. The complexity is estimated as

$$T_{D^*} = (17 \times 32)^3 \times 2^{2 \times 35} \times 2^{3 \times 27} \times 2^{2 \times 19} \times 2^{34 \times 3} = (17 \times 32)^3 \times 2^{291}.$$

This means that our algorithm gives the tap selection with much better resistance against GFSGA.

SFINX: The design details of SFINX can be found in [19]. The set of the tap positions of SFINX is given as

$$\mathcal{I}_0 = \{1, 2, 7, 10, 20, 22, 45, 59, 75, 99, 106, 135, 162, 194, 228, 245, 256\},$$

and $D = \{1, 5, 3, 10, 2, 23, 14, 16, 24, 7, 29, 27, 32, 34, 17, 11\}$. An optimal step of the GFSGA attack on this set of tap positions, is $\sigma = 2$ which requires $c = 27$ sampling points, resulting in $R = 200$ sampled equations for obtaining an overdefined system. The corresponding complexity in this case is $T_{Comp.} = 2^{256}$. Note that $\sum_{i=1}^{16} d_i = 255$ with optimal step $\sigma = 2$, which indicates that the set of tap positions \mathcal{I}_0 of SFINX is chosen well. However, we can use the elements of the given set D and our algorithm to create the set of differences “by parts”, in order to decrease the number of repeated equations R and increase the complexity (slightly). Starting with the set $X = \{29, 32, 17, 34, 27, 11\}$, and permuting the set $Y_p = \{2, 23, 14, 16, 24, 7\}$ for $L_{Y_p, X} = 237$, we get the set

$Y_p X = \{2, 23, 14, 7, 16, 24, 29, 32, 17, 34, 27, 11\}$ with an optimal step $\sigma = 8$ for the attack. Then, taking the set $Z_p = \{1, 5, 3, 10\}$, we get the set $D^* = Z_p Y_p X$ given as

$$D^* = \{1, 5, 3, 10, 2, 23, 14, 7, 16, 24, 29, 32, 17, 34, 27, 11\},$$

with the optimal steps $\sigma \in \{1, 2\}$ for the attack. The estimated complexity for both optimal steps is $T_{Comp.} = 2^{257}$ with $R = 167$ repeated equations, thus only a minor improvement has been achieved.

It would be of interest to consider the problem of optimizing the placement of tap positions in case the GFSGA attack with a variable sampling step (σ is not fixed) is used, which is left for the extended version of this article.

Acknowledgments. Enes Pasalic was supported in part by the Slovenian Research Agency research program (P3-0384) and research project (J1-6720). Samir Hodžić was supported in part by the Slovenian Research Agency (research program P3-0384 and Young Researchers Grant). Yongzhuang Wei was supported in part by the Natural Science Foundation of China (61100185, 61201250), in part by the National Basic Research Program of China (2013CB338002), in part by the project of Outstanding Young Teachers' Training in Higher Education Institutions of Guangxi.

Appendix

In Table 5 we give several instances for determining suboptimal tap positions of LFSRs of different length. The following parameters are used:

- L is the length of LFSR;
- n and m are parameters related to vectorial Boolean function $F : GF(2)^n \rightarrow GF(2)^m$;
- D is a set of differences between tap positions;
- c is the minimal number of observed outputs needed for an overdefined system
- R is the number of repeated equations for given c outputs;
- σ is an optimal step of the GFSGA attack;
- $T_{Comp.}$ is the time complexity of GFSGA.

Table 5. Specifications of difference sets for LFSRs of different lengths.

L	(n, m)	D	R	c	σ	$T_{Comp.}$
80	(7,2)	$\{5, 13, 7, 26, 11, 17\}$	24	15	1	$2^{69.97}$
120	(13,3)	$\{5, 7, 3, 13, 6, 11, 5, 11, 7, 13, 21, 17\}$	61	14	3	$2^{99.7}$
160	(17,6)	$\{5, 11, 4, 3, 7, 9, 1, 2, 23, 15, 5, 13, 7, 26, 11, 17\}$	128	17	1	$2^{86.97}$
200	(21,7)	$\{3, 7, 9, 13, 18, 7, 9, 1, 2, 9, 1, 2, 23, 15, 5, 13, 7, 26, 11, 17\}$	175	18	1	$2^{108.9}$
256	(27,9)	$\{5, 9, 13, 4, 7, 19, 3, 7, 9, 13, 18, 7, 9, 1, 2, 9, 1, 2, 23, 15, 5, 13, 7, 26, 11, 17\}$	227	18	1	2^{135}

Remark 7. From the difference sets D in Table 5 we easily obtain the tap positions.

Table 6. Time complexities for finding tap positions in Table 5.

L	(n, m)	Cardinality of parts	Complexity	Times in <i>sec</i>
80	(7,2)	no parts	$O(K \cdot 6!)$	135
120	(13,3)	(6,6)	$2 \cdot O(K \cdot 6!)$	125+162=287
160	(17,6)	(6,6,4)	$2 \cdot O(K \cdot 6!) + O(K \cdot 4!)$	137+198+8.5=343.5
200	(21,7)	(6,6,4,4)	$2 \cdot O(K \cdot 6!) + 2 \cdot O(K \cdot 4!)$	137+96+7.7+9.5=250
256	(27,9)	(6,6,4,4,6)	$3 \cdot O(K \cdot 6!) + 2 \cdot O(K \cdot 4!)$	250+369.3=619.3

Remark 8. Note that the time required to create some particular set of differences depends on the cardinality of parts. It means that the smaller cardinalities implies the lower time complexity, though such an approach may provide the solutions that are “far” from optimal. Table 6 presents the following:

- Cardinality of parts refers to the modified algorithm on Page 10, bottom. For instance, (6,6,4) means that we take $\#X = 6$ elements and finding its optimal permutation requires 137 sec with our permutation algorithm. Then, we take another $\#Y_p = 6$ elements and determine its best order which fits to the set X , which requires 198 seconds (modified algorithm). Finally, the same procedure is applied to the set $Y_p X$ by adding $Z_p = 4$ elements using again our modified algorithm (requiring 8.5 sec). The resulting set of differences is given as $D = Z_p Y_p X$.
- Complexity refers to the complexity of the permutation algorithms **Step B** and its modification used to construct the set D .
- The constant K regards the procedure described in the permutation algorithm (**Step B**): creating the list, searching, etc.

References

1. Biryukov, A., Shamir, A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 1–13. Springer, Heidelberg (2000)
2. Braeken, A., Preneel, B.: Probabilistic algebraic attacks. In: Smart, N.P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 290–303. Springer, Heidelberg (2005)
3. Carlet, C.: A larger class of cryptographic boolean functions via a study of the Maiorana-McFarland construction. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 549–564. Springer, Heidelberg (2002)
4. Courtois, N., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: Boneh, D. (ed.) Advances in Cryptology-EUROCRYPT 2003. LNCS, vol. 2656, pp. 346–359. Springer, Heidelberg (2003)

5. Golić, J.D.: Intrinsic statistical weakness of keystream generators. In: Pieprzyk, J., Safavi-Naini, R. (eds.) *Advances in Cryptology-ASIACRYPT 1994*. LNCS, vol. 917, pp. 91–103. Springer, Heidelberg (1995)
6. Golić, J.D.: On the security of nonlinear filter generators. In: Gollmann, D. (ed.) *Fast Software Encryption 1996*. LNCS, vol. 1039, pp. 173–188. Springer, Heidelberg (1996)
7. Golic, J.D., Clark, A., Dawson, E.: Generalized inversion attack on nonlinear filter generators. *IEEE Trans. Comput.* **49**(10), 1100–1109 (2000)
8. Hellman, M.: A cryptanalytic time-memory tradeoff. *IEEE Trans. on Inform. Theor.* **26**(4), 401–406 (1980)
9. Hong, J., Sarkar, P.: New applications of time memory data tradeoffs. In: Roy, B. (ed.) *ASIACRYPT 2005*. LNCS, vol. 3788, pp. 353–372. Springer, Heidelberg (2005)
10. Massey, J.L.: Shift-register synthesis and BCH decoding. *IEEE Trans. Inform. Theor.* **15**(1), 122–127 (1969)
11. Meier, W., Pasalic, E., Carlet, C.: Algebraic attacks and decomposition of Boolean functions. In: Cachin, C., Camenisch, J.L. (eds.) *EUROCRYPT 2004*. LNCS, vol. 3027, pp. 474–491. Springer, Heidelberg (2004)
12. Mihaljević, M.J., Fossorier, M.P.C., Imai, H.: A general formulation of algebraic and fast correlation attacks based on dedicated sample decimation. In: Fossorier, M.P.C., Imai, H., Lin, S., Poli, A. (eds.) *AAECC 2006*. LNCS, vol. 3857, pp. 203–214. Springer, Heidelberg (2006)
13. Mihaljević, M.J., Gangopadhyay, S., Paul, G., Imai, H.: Internal state recovery of Grain-v1 employing normality order of the filter function. *IET Inform. Secur.* **6**(2), 55–64 (2006)
14. Nyberg, K.: On the construction of highly nonlinear permutations. In: Rueppel, R.A. (ed.) *EUROCRYPT 1992*. LNCS, vol. 658, pp. 92–98. Springer, Heidelberg (1993)
15. Pasalic, E.: On guess and determine cryptanalysis of LFSR-based stream ciphers. *IEEE Trans. Inform. Theor.* **55**(7), 3398–3406 (2009)
16. Hawkes, P., Rose, G.: Primitive specification and supporting documentation for SOBER-t16 submission to NESSIE. In: *Proceedings of the First Open NESSIE Workshop*, KU-Leuven (2000)
17. Pasalic, E.: Probabilistic versus deterministic algebraic cryptanalysis performance comparison. *IEEE Trans. Inform. Theor.* **55**(11), 2182–2191 (2009)
18. Wei, Y., Pasalic, E., Hu, Y.: Guess and determinate attacks on filter generators-revisited. *IEEE Trans. Inform. Theor.* **58**(4), 2530–2539 (2012)
19. Braeken, A., Lano, J., Mentens, N., Preneel, B., Verbauwhede, I.: SFINKS: A Synchronous Stream Cipher for Restricted Hardware Environments. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/026 (2005)

Cryptography and Information Security in the Balkans
First International Conference, BalkanCryptSec 2014,
Istanbul, Turkey, October 16-17, 2014, Revised
Selected Papers
Ors, B.; Preneel, B. (Eds.)
2015, X, 251 p. 43 illus., Softcover
ISBN: 978-3-319-21355-2