Analysis of the Bit-Search Generator and sequence compression techniques*

Aline Gouget[1], Hervé Sibert[1], Côme Berbain[2], Nicolas Courtois[3], Blandine Debraize[3,4], and Chris Mitchell[5]

[1] France Telecom Research and Development,
42 rue des Coutures, F-14000 Caen, France.
[2] France Telecom Research and Development,
38-40 rue du Général Leclerc, F-92794 Issy-les-Moulineaux, France.
[3] Axalto Cryptographic Research & Advanced Security,
36-38 rue de la Princesse, BP 45, F-78430 Louveciennes Cedex, France.
[4] Versailles University, 45 avenue des Etats-Unis, F-78035 Versailles, France.
[5] Information Security Group, Royal Holloway, University of London,
Egham, Surrey TW20 0EX, United Kingdom

Abstract. Algebraic attacks on stream ciphers apply (at least theoretically) to all LFSR-based stream ciphers that are clocked in a simple and/or easily predictable way. One interesting approach to help resist such attacks is to add a component that de-synchronizes the output bits of the cipher from the clock of the LFSR. The Bit-search generator, recently proposed by Gouget and Sibert, is inspired by the so-called Self-Shrinking Generator which is known for its simplicity (conception and implementation-wise) linked with some interesting properties. In this paper, we introduce two modified versions of the BSG, called MBSG and ABSG, and some of their properties are studied. We apply a range of cryptanalytic techniques in order to compare the security of the BSGs.

1 Introduction

In recent years there has been renewed interest in designing stream cipher keystream generators (KGs) capable of being implemented in small software or hardware and operating at very high rates. The *Shrinking Generator (SG)* [2] and *Self-Shrinking Generator (SSG)* [8] are schemes providing a method for irregular decimation of pseudorandom sequences such as those generated by linear feedback shift registers (LFSRs).

Recently, Gouget and Sibert [6] introduced the *Bit-Search Generator* (BSG), that is, like the SG and SSG, a scheme designed to offer attractive characteristics for both software and hardware implementation when used as a part of a KG. However, similarly to the SG and the SSG, the BSG can be vulnerable to timing attacks. The BSG has the advantage over the SG and SSG that it operates at a

* Work partially supported by the French Ministry of Research RNRT Project "X-CRYPT" and by the European Commission via ECRYPT network of excellence IST-2002-507932.

Appeared in Henri Gilbert and Helena Handschuh(Eds.): FSE 2005, LNCS 3557,pp. 196–214, 2005.

rate of 1/3 instead of 1/4 (i.e. producing n bits of the output sequence requires, on average, $3n$ bits of the input sequence).

Given that the BSG is aimed as a building block for constructing a KG, it is essential to know how simple it is to reconstruct parts of the input sequence from the output. This arises naturally in the context of stream cipher design, where matching known plaintext and ciphertext immediately gives keystream values, i.e. subsequences of the output sequence, and where knowledge of parts of the input sequence is a prerequisite to determining the secret key used to generate the sequence. Furthermore, in order to avoid algebraic attacks (see among other [1, 3]), it is important to know how many relations that relate some outputs bits to consecutive input bits can be obtained.

The outline of the paper is as follows. In Section 2, we recall the original description of the BSG and we provide an equivalent specification which operates on the *differential* of the original sequence. In Section 3, we consider two strategies in order to reconstruct the original sequence from the output sequence of the BSG. We give a basic attack which has complexity $\mathcal{O}(L^3 2^{\frac{L}{3}})$ and requires $\mathcal{O}(L 2^{\frac{L}{3}})$ keystream bits, where L is the length of the underlying LFSR. We then improve this attack to get a complexity of $\mathcal{O}(L^3 2^{\frac{L}{4}})$. In Section 4, we propose two modified versions of the BSG designed to increase its security. Analogously to the work in [6] for the BSG, we study some properties of both the MBSG and the ABSG. In Section 5, we apply, to both the MBSG and the ABSG, the strategies of Section 3 and the FBDD attack against LFSR-based generators introduced by Krause in [7]. The best attack that we give against the ABSG and the MBSG has complexity $\mathcal{O}(2^{\frac{L}{2}})$ and requires $\mathcal{O}(L 2^{\frac{L}{2}})$ bits of keystream. Finally, we conclude in Section 6.

2 The Bit-Search Generator

One can consider that both the SG and SSG are methods for *bit-search*-based decimation. Indeed, both generators use a search for ones along an input bit sequence in order to determine the output bit. Instead of using a search for ones, the BSG uses a search for some bit b, where b varies during the process; the variations depend on the input sequence. During the search process for a bit b, a cursor moves along the input sequence. The search process ends when the next occurrence of b is reached. Then, the output bit is zero if the search process ends after just reading one bit, otherwise the output is one. The next value of the bit b corresponds to the value of the following bit of the sequence.

We recall the original description of the BSG given in [6] and we provide an equivalent specification of the BSG which operates on the *differential* of the original sequence; the *differential sequence* $d = (d_0, d_1, \dots)$ of a sequence s is defined by $d_i = s_i \oplus s_{i+1}$, $i \geq 0$, where $\oplus$ denotes bit-wise exclusive-or (or modulo 2 addition). As usual, the complement of b in $\{0, 1\}$ is denoted $\overline{b}$.

Definition 1 (BSG). *Let $s = (s_0, s_1, \dots)$ be a pseudorandom bit sequence and $d = (d_0, d_1, \dots)$ be the differential sequence. The output sequence $y = (y_0, y_1, \dots)$ of the BSG is constructed as follows:*

Appeared in Henri Gilbert and Helena Handschuh(Eds.): FSE 2005, LNCS 3557,pp. 196–214, 2005.

```
BSG (original)
Input: $(s_0, s_1, \dots)$
Set: $i \leftarrow 0$; $j \leftarrow 0$;
Repeat the following steps:
  1. $e \leftarrow s_i$, $y_j \leftarrow s_i \oplus s_{i+1}$;
  2. $i \leftarrow i + 1$;
  3. while $(s_i = \overline{e})$ $i \leftarrow i + 1$;
  4. $i \leftarrow i + 1$;
  5. output $y_j$;
  6. $j \leftarrow j + 1$;
```

```
BSG_diff (differential)
Input: $(d_0, d_1, \dots)$
Set: $i \leftarrow 0$; $j \leftarrow 0$;
Repeat the following steps:
  1. $y_j \leftarrow d_i$;
  2. if $(y_j = 1)$ then
       (a) $i \leftarrow i + 1$;
       (b) while $(d_i = 0)$ $i \leftarrow i + 1$;
  3. $i \leftarrow i + 2$;
  4. output $y_j$;
  5. $j \leftarrow j + 1$;
```

Example 1. Let $s = 010100111010010001 1101$ be a bit sequence. Then, the action of the BSG on s is described by:

$$\underbrace{010}_{1}\underbrace{1001}_{1}\underbrace{11}_{0}\underbrace{010}_{1}\underbrace{010}_{1}\underbrace{00}_{0}\underbrace{11}_{0}\underbrace{101}_{1} \quad .$$

The action of the BSG on the input sequence s consists in splitting up the sequence s into subsequences of the form $(\bar{b}, b^i, \bar{b})$ where $b \in \{0, 1\}$ and $i \geq 0$. For every subsequence of the form $(\bar{b}, b^i, \bar{b})$, the output bit is 0 if $i = 0$, and 1 otherwise. The action of the BSG on the input differential sequence d consists in splitting up the subsequence d into subsequences of the form either $(0, b)$ or $(1, 0^i, 1, b)$ with $i \geq 0$; for every such subsequence, the output bit is the first bit of the subsequence.

It is simple to verify that both descriptions of the BSG are equivalent given that the output bit is zero when the search along the sequence s ends immediately and it is one otherwise. We denote the output sequence of the BSG by $BSG(s)$ or $BSG_{\text{diff}}(d)$ depending on the sequence we are focusing on.

Remark 1. Recovering elements of the sequence d is likely to be of very similar signifiance to recovering elements of s. For instance, when s is generated using an LFSR, then d can also be generated using an identical LFSR [5, 9]. Furthermore, the transformation from s to d simply shifts the position of the starting point of the sequence. In this case, recovering the entire sequence d from partial information has precisely the same difficulty as for the sequence s.

Assuming that the input sequence of the BSG is evenly distributed, then the output rate of the BSG is clearly 1/3 (the number of input bits required to produce one output bit is $1 + i$ with probability $1/2^i$ $(i \geq 1)$).

Proposition 1. *Assume that the output sequence y produced by the BSG is evenly distributed. Then, for each output bit y_j, the expected number of known input bits is* 2 *with an average entropy of* 1.

Proof. Every zero in y corresponds to a pair of bits $(0, b)$ in the differential sequence d of the original sequence s, and no information is available about the bit b. Thus, if an output bit is a zero, then one input bit of d is known.

Every one in y corresponds to a pattern $(1, 0^i, 1, b)$ with $i \geq 0$ in d and the following possibilities exist: two bits are known with probability $1/2$, three bits are known with probability $1/4, \ldots$, that is $1+i$ bits are known with probability $1/2^i$ for $i \geq 1$. Hence the expected number of known bits is $\sum_{i=1}^{\infty}(1+i)/2^i = 3$. The associated entropy is given by $\sum_{i=1}^{\infty} 2^{-i} \log_2(2^{-i}) = \sum_{i=1}^{\infty} i2^{-i} = 2$. Thus, assuming that the output sequence is evenly distributed, for each output bit the expected number of known input bits is 2, with an average entropy of 1. □

3 How to Reconstruct the Input Sequence?

In this section, we consider two approaches, called *Strategy 1* and *Strategy 2* in order to evaluate how simple it is to reconstruct parts of either the input sequence s or its differential from the output sequence y.

For the first approach, called *Strategy 1*, we assume that we have no additionnal information on the means used to generate the input sequence. This approach is based on the random generation of *candidates* for the input sequence which are *consistent* with the information derived from the output sequence. For the second approach, called *Strategy* 2, we assume that the feedback polynomial used to generate the input sequence is known. This second approach consists of building an attack on the BSG based on the choice of the most probable case for LFSR sequences as input.

3.1 Strategy 1: Use of Random Generation of Candidates

Consider a bit sequence s, its differential sequence d and the output sequence $y = BSG(s) = BSG_{\mathrm{diff}}(d)$. In this approach, we focus on the reconstruction of the differential sequence d that we call the *correct input string* and we assume that we have no additional information on the means used to generate the input sequence.

A sequence c is called a *differential-candidate* for the output sequence y if the equality $BSG_{\mathrm{diff}}(c) = y$ is fulfilled. One method to search for the correct input string is to randomly generate a sequence of differential candidates for the input bits. The probability of success of such a strategy depends on the Hamming weight w of the subsequence, i.e. there are w places in the input sequence where a string of zeros of uncertain length may occur. Recall that every one in the output sequence arises from a tuple of the form $(1, 0^i, 1, b)$, where $i \geq 0$ and b is an undetermined bit. The Hamming weight of a finite sequence y is denoted by $w(y)$.

Proposition 2. *Let d be a (finite) bit sequence and y be a sequence such that $y = BSG_{diff}(d)$. Let c be a randomly chosen string with the property that $BSG_{diff}(c) = y$, where the probability distribution used to choose c reflects the probability that $c = d$. The probability that, for every k such that $y_k = 1$, the sequences d and c agree on the length of the tuple from which y_k arises, is $3^{-w(y)}$.*

Proof. Each differential-candidate input string should have a tuple $(1, 0^i, 1)$ inserted for every one occurring in the output sequence; i is chosen independently at random for each output bit and $i = j$ with probability 2^{-j-1}. In each of the $w(y)$ locations, a string of i zeros occurs in the correct input sequence with probability 2^{-i-1}. The probability that the candidate string and the correct string agree in any one of the $w(y)$ positions is thus $\sum_{i=0}^{\infty}(2^{-i-1})^2 = 1/3$. That is, the probability that the correct input sequence and a candidate c agree on the $w(y)$ choices of length of the tuples from which the ones of y arises is $3^{-w(y)} \simeq 2^{-1.585y}$. □

Thus, finding one output sequence with small Hamming weight yields attacks that are likely to be easier than brute force attacks. This idea is used in Strategy 2.

3.2 Strategy 2: Choice of the Most Probable Case

The goal of Strategy 2 is the reconstruction of the original input sequence s. We assume that s is generated by a maximum length LFSR of size L with a public feedback polynomial and the initial state of the LFSR is the secret key. We further suppose that the feedback polynomial has been chosen carefully, i.e. it does not have a low Hamming weight and no low weight multiple exists, in order to avoid attacks on the differential sequence similar to the distinguishing attack on the SG given in [4].

Recall that each zero of the output sequence y comes from two consecutive equal bits in the input sequence s. Thus, each zero in y provides a linear equation over the unknown LFSR sequence, namely the equality between two consecutive bits. Similarly, each one of y comes from a pattern $(\bar{b}, b^i, \bar{b})$ for some integer $i \geq 1$. Thus, by guessing i, we can construct $i+1$ linear equations involving consecutive bits of the unknown LFSR sequence which are valid with probability 2^{-i}.

Basic attack Let us take the first window of $2L/3$ consecutive bits in the sequence y with a Hamming weight of at most $L/3$. For a random window of size $2L/3$, this condition is satisfied with probability close to $1/2$, so that the first window can be found in negligible time. If the Hamming weight of the window is strictly lower than $L/3$, we expand it in such a way that it contains exactly $L/3$ ones (or until its size is L). We now assume that each one in the sequence y comes from a pattern of length 3, that is a pattern of the form $\bar{b}b\bar{b}$, which is the most probable case, occuring with probability $2^{-\frac{L}{3}}$. Then, we can write L equations involving consecutive bits of the LFSR sequence or, equivalently, the bits of the current state. We solve this system and we instantly check if we have found the correct values by testing whether it allows to the correct prediction of a few additional bits of the sequence y. In order to find the current state with high probability (close to $1 - \frac{1}{e}$), we have to repeat this procedure $2^{\frac{L}{3}}$ times. This attack costs $\mathcal{O}(L^3 2^{\frac{L}{3}})$ and requires $\mathcal{O}(L2^{\frac{L}{3}})$ bits of keystream.

Improvements to the basic attack We tried several alternative strategies such as finding a large enough keystream window with a low Hamming weight, or connecting two smaller windows of low weight. For instance, we can determine, in a first computation phase, 2^w windows of size ℓ bits and Hamming weight w. For each of these windows, we suppose that every one comes from a pattern $(\bar{b}, b, \bar{b})$, which gives $\ell + w$ linear equations. These equations are all valid with probability 2^{-w}. This costs:

$$\mathcal{O}\left(\frac{2^{w+\ell+1}}{\binom{\ell}{w}}\right).$$

For each pair of such windows, we know the number n_1 of ones and n_0 of zeros in the sequence y between the two windows. Considering all the possible strings $\bar{b}b^i\bar{b}$ for integer $i \geq 1$, the mean value m of i and the variance v are given by:

$$m = \sum_{k=1}^{\infty} \frac{k}{2^k} = 2 \ , \qquad v = \sum_{k=1}^{\infty} \frac{(k-2)^2}{2^k} = 2.$$

Thus, the distance between those two windows in the original sequence is likely to belong to the interval $[2n_0+4n_1-\sqrt{2n_1}, 2n_0+4n_1+\sqrt{2n_1}]$. The Central Limit Theorem gives the probability that the real distance between the two windows is outside this interval:

$$Pr\left(\frac{\sum_{i=1}^{n_1} X_i - mn_1}{\sqrt{vn_1}} \geq 1\right) = \frac{2}{\sqrt{2\pi}} \int_1^{\infty} e^{-\frac{x^2}{2}} dx \simeq 0.31.$$

Therefore, for each pair of windows, the probability of failure provided that the distance used is not correct is around 1/3. We try all the values of the distance between the two windows in this interval. If we make a correct guess, the equations associated to the two windows can be combined to provide $2(\ell+w)$ equations. We choose ℓ and w such that $2(\ell+w) = L$ and we just have to solve the system so as to test whether the obtained solution correctly predicts a few additional keystream bits.

Since n_1 is $\mathcal{O}\left(\frac{2^w 2^\ell}{\binom{\ell}{w}}\right)$, testing all the pairs of windows costs $\mathcal{O}\left(2^{2w+1}\sqrt{2\frac{2^w 2^\ell}{\binom{\ell}{w}}}\right)$, and the total complexity of the attack is:

$$\mathcal{O}\left(\frac{2^{w+\ell+1}}{\binom{\ell}{w}} + 2^{2w+1}\sqrt{2\frac{2^{w+\ell}}{\binom{\ell}{w}}}L^3\right).$$

Moreover the number of keystream bits required for the attack is:

$$\mathcal{O}\left(\ell 2^w \frac{2^\ell}{\binom{\ell}{w}}\right).$$

For practical values of L ($L \in [128, 4096]$), $\ell = \frac{25L}{58}$ and $w = \frac{7L}{116}$, this provides a complexity close to or slightly smaller than $L^3 2^{\frac{L}{4}}$ and a keystream length of $2^{\frac{L}{4}}$.

4 New BSGs to Improve the Security?

The discussion in section 3 suggests that the security of the BSG relies on the uncertainty about the length of the input tuple required to output a one. By contrast, if a zero is output, then there is no uncertainty about the length of the input string. This suggests that the security might be improved by introducing ambiguity no matter whether a zero or a one is output by the scheme.

Remark 2. Instead of aiming at an improvement in security, one may want to enhance the rate with the same level of security. Indeed, a simple modification to the BSG enables its rate to be increased from 1/3 to 1/2 by changing Step 3 in the BSG_{diff} Algorithm (Definition 1) from $i \leftarrow i+2$ to $i \leftarrow i+1$. However, an adaptation of the basic attack presented in Section 3.2 to this case (for an LFSR input) leads to an attack which costs $\mathcal{O}(2^{\frac{L}{3}})$ and requires $\mathcal{O}(L2^{\frac{L}{3}})$ bits of keystream; the security is then slightly lower than for the BSG.

4.1 BSG Variants

We give two possible modifications of the BSG that are called the MBSG and the ABSG; these two modifications are not equivalent (even if we consider the differential sequence instead of the original sequence).

Definition 2 (MBSG & ABSG). *Let $s = (s_0, s_1, \dots)$ be a pseudorandom bit sequence. The output sequences of the MBSG and of the ABSG are constructed as follows.*

MBSG algorithm	*ABSG algorithm*
Input: $(s_0, s_1, \dots)$	*Input:* $(s_0, s_1, \dots)$
Set: $i \leftarrow 0$; $j \leftarrow 0$;	*Set:* $i \leftarrow 0$; $j \leftarrow 0$;
Repeat the following steps:	*Repeat the following steps:*
1. $y_j \leftarrow s_i$;	*1.* $e \leftarrow s_i$, $y_j \leftarrow s_{i+1}$;
2. $i \leftarrow i+1$;	*2.* $i \leftarrow i+1$;
3. while $(s_i = 0)$ $i \leftarrow i+1$;	*3. while* $(s_i = \overline{e})$ $i \leftarrow i+1$;
4. $i \leftarrow i+1$;	*4.* $i \leftarrow i+1$;
5. output y_j;	*5. output* y_j
6. $j \leftarrow j+1$;	*6.* $j \leftarrow j+1$

Example 2. Let $s = 010100111010010001110 1$ be the input bit sequence. Then, the action of the MBSG on s is described by:

$$\underbrace{01}_{0}\underbrace{01}_{0}\underbrace{001}_{0}\underbrace{11}_{1}\underbrace{01}_{0}\underbrace{001}_{0}\underbrace{0001}_{0}\underbrace{11}_{1}\underbrace{01}_{0} \quad ,$$

and the action of the ABSG on s is described by:

$$\underbrace{010}_{1}\underbrace{1001}_{0}\underbrace{11}_{1}\underbrace{010}_{1}\underbrace{010}_{1}\underbrace{00}_{0}\underbrace{11}_{1}\underbrace{101}_{0} \quad .$$

The action of the MBSG on the input sequence s consists in splitting up s into subsequences of the form $(b, 0^i, 1)$, with $i \geq 0$ and $b \in \{0, 1\}$. For every pattern of the form $(b, 0^i, 1)$, the output bit is b. The action of the ABSG on s consists in splitting up s into subsequences of the form $(\bar{b}, b^i, \bar{b})$, with $i \geq 0$ and $b \in \{0, 1\}$. For every subsequence $(\bar{b}, b^i, \bar{b})$, the output bit is b for $i = 0$, and $\bar{b}$ otherwise. Both the MBSG and the ABSG clearly have a rate of 1/3, like the BSG. Indeed, for every $i \geq 1$, an output bit is produced by $1 + i$ input bits with probability $1/2^i$.

Remark 3. The action of the ABSG on an input sequence is identical to that of the BSG, but their outputs are computed differently.

Proposition 3. *Let s be a pseudorandom bit sequence. Assume that the output sequence $y = MBSG(s)$ is evenly distributed. Then for every output bit y_j, the expected number of known bits of s is* 3 *with an average entropy of* 2.

Proof. If an output bit is a b, then the input sequence used to generate this output bit must have the form $(b, 0^i, 1)$, where $i \geq 0$ and $i = j$ with probability 2^{-j-1}. Thus, if an output bit is a b, then $i + 1$ bits are known with probability $1/2^i$ for $i \geq 1$. As shown in the proof of Proposition 1, the expected number of known bits is 3 and the associated entropy is 2. □

Proposition 3 also holds for the ABSG.

4.2 Filtering Periodic Input Sequences

We now describe the output of the MBSG and ABSG when applied to periodic sequences (of period greater than 1) as was done in [6] for the BSG. We will show that the BSG and the ABSG on the one hand, and the MBSG on the other hand, behave differently in this regard.

Definition 3. *For two sequences $s = (s_i)_{i \geq 0}$ and $s' = (s'_i)_{i \geq 0}$, we say that s' is* (k-)shifted *from s if there exists $k \geq 0$ such that $s'_i = s_{i+k}$ for every $i \geq 0$.*

As usual, s is said to be *eventually periodic* if there exists a shifted sequence from s which is periodic. We denote by $BSG(s, i)$ (resp. $MBSG(s, i)$, $ABSG(s, i)$) the i-shifted sequence from $BSG(s)$ (resp. $MBSG(s)$, $ABSG(s)$).

The next proposition was proved in [6] for the BSG. It also holds for the ABSG thanks to the fact that the ABSG acts like the BSG on the input sequence.

Proposition 4. *Let s be a sequence of period T. Then, the sequence $ABSG(s)$ is periodic, and there exists $k \in \{1, 2, 3\}$ such that $ABSG(s_0, \ldots, s_{kT-1})$ is a period of $ABSG(s)$.*

The framework introduced in [6] uses the associated permutation p to a periodic sequence s: we define two transpositions $t_0 = (\emptyset\ 0)$ and $t_1 = (\emptyset\ 1)$. Then, we associate with s the permutation $t_{s_{T-1}} \circ \cdots \circ t_{s_0}$ over the set $\{\emptyset, 0, 1\}$. The integer k in the previous proposition is the order of the permutation associated with s.

For the MBSG, the picture is slightly different. The MBSG acts on an input sequence s as follows: read a bit b, go to the next occurrence of one and start again. We give to the cursor moving along the input sequence two states: $\emptyset$ when there is no current bit looked for, and 1 otherwise. The cursor changes from state $\emptyset$ to state 1 after reading a bit. When the cursor is in state 1, it remains in state 1 if the next bit read is 0, and changes to state $\emptyset$ if the next bit read is 1.

Proposition 5. *Let s be a sequence of period T. Then, the sequence $MBSG(s)$ is eventually periodic. Moreover, if $s_{i-1}s_i$ is an occurrence of $(0,1)$ in s, then the sequence $MBSG(s, i+1)$ is periodic and a period is $MBSG(s_{i+1}, \ldots, s_{i+T})$.*

Proof. After reading a pattern $(0,1)$, the cursor is always in state $\emptyset$, and thus the bit s_i is the last bit read during some search process. Now, as the cursor is in state $\emptyset$ after s_i, it will also be in this state after s_{i+kT} for every k. □

In the sequel, we denote by $MBSG_P(s)$ the sequence $MBSG(s, i+1)$ where (s_{i-1}, s_i) is the first occurrence of $(0,1)$ in s. Thus $MBSG_P(s)$ is a periodic shift of $MBSG(s)$.

Output Sequence Sets and Shifts. Given an input sequence s of period T, one can filter the shifted sequences (s,i) for $0 \leq i \leq T-1$, so as to obtain at most T distinct output sequences. We call the set of these output sequences the *output sequence set for input s*. We will show that these output sequences are closely related to one another. The following proposition was proved in [6] in the case of the BSG using only the action of the BSG on the input sequence. Thus, it also holds for the ABSG.

Proposition 6. *Let $s = (s_i)_{i \geq 0}$ be an infinite bit sequence and k be the minimal index such that $s_k \neq s_0$. Then, for every $i \geq 0$, the sequence $ABSG(s,i)$ is shifted from one sequence among $ABSG(s,0)$, $ABSG(s,1)$ and $ABSG(s, k+1)$.*

In the case of the MBSG, we have to consider the periodic part $MBSG_P(s)$ so as to obtain a similar proposition:

Proposition 7. *Let $s = (s_i)_{i \geq 0}$ be an infinite bit sequence where both 0 and 1 appears infinitely many times. Then, for every $i \geq 0$, the sequence $MBSG_P(s,i)$ is shifted from the sequence $MBSG_P(s)$.*

Proof. Let us consider the cursor in initial state $\emptyset$ running along the sequence s. Let $s_{k-1}s_k$ be the first occurrence of 01 in the sequence (s,i). After reading a pattern $(0,1)$, the cursor is always in state $\emptyset$. Thus, the cursor is in state $\emptyset$ after reading s_k. Therefore, $MBSG(s, k+1)$ is shifted from both $MBSG(s)$ and $MBSG(s,i)$. Now, $MBSG(s, k+1)$ is periodic, which yields the result. □

Maximum Length LFSR Sequences as Input. When the input sequence s is produced by a maximum length LFSR, the periodicity properties differ between the MBSG on the one hand, and the BSG and the ABSG on the other hand.

Appeared in Henri Gilbert and Helena Handschuh(Eds.): FSE 2005, LNCS 3557,pp. 196–214, 2005.

A lower bound on the length of $BSG(s_0, \ldots, s_{kT-1})$, where $k \in \{1, 2, 3\}$, is the order of the permutation associated with s, was proven in [6]. The proof also holds for the ABSG:

Proposition 8. *[6] Suppose s is the output of a maximum length LFSR of degree $L \geq 3$, and let p be the associated permutation. Let k be the minimal strictly positive integer such that $p^k(\emptyset) = \emptyset$. The length of the sequences $BSG(s_0, \ldots, s_{kT-1})$ and $ABSG(s_0, \ldots, s_{kT-1})$ are both greater than $k \cdot 2^{L-3}$.*

This bound does not answer the issue of possible subperiods. A strict lower bound on the period length of $BSG(s)$ was introduced in [6]. Experimentally, for both the BSG and the ABSG, no subperiod appears when the input is produced by a maximal-length LFSR with feedback polynomial of degree $3 \leq L \leq 16$. As was done in [6] for the BSG, one can show that the output sequence set of the ABSG can be easily described from 2 distinct output sequences whose period lengths, called *short period* and *long period*, are respectively, when no subperiod appears, very close to $T/3$ and $2T/3$, and their sum is then exactly T. The results for the ABSG are given in Tables 2 of Appendix C. For the MBSG, we have:

Proposition 9. *Let s be a sequence produced by a maximum length LFSR of degree L. Consider a period of the output of the form $0^{\lambda_1}1^{\mu_1}0^{\lambda_2}1^{\mu_2}\ldots 0^{\lambda_p}1^{\mu_p}$. Then, the sequence $MBSG_P(s)$ has a period $MBSG(t)$ of length T such that:*

- *for $L = 0 \mod 2$, we have $T = (2^L - 1)/3$, the number of zeros in this period is $(T-1)/2$, and the number of ones is $(T+1)/2$,*
- *for $L = 1 \mod 2$, we have $T = (2^L + 1)/3$, the number of zeros in this period is $(T+1)/2$, and the number of ones is $(T-1)/2$.*

The proof of Proposition 9 is given in Appendix A.

Like for the BSG and the ABSG, subperiods may appear in a period of $MBSG_P(s)$ of length T. Experimentally, this never happens for $L \leq 16$, so that the values in Proposition 9 are exact. The periodicity results are given in Table 4 in Appendix C.

Linear Complexity of Output Sequences. We do not have theoretical bounds for the linear complexity, but the statistics for maximum length LFSRs of degree $L \leq 16$ suggest that the linear complexity is well-behaved. The results for the linear complexity are given in Appendix C, in Tables 3 and 4 respectively for the ABSG and the MBSG.

For the ABSG, we give the average linear complexity (denoted by LC), and its minimal and maximal values for short and long output sequences. These values are to be compared with those in Table 2: indeed, they show that the linear complexity is always almost equal to the period.

For the MBSG, preliminary experiments on the linear complexity of the output sequences when filtering maximum length LFSR sequences show that the linear complexity is very close to the period. Furthermore, when the period is prime, we observe that the linear complexity is always equal to the period (for degrees up to 16, for which we tested every possible maximum length LFSR output). Therefore, further study of the MBSG seems promising.

5 Security of the MBSG and the ABSG

5.1 Strategy 1: random generation of candidates

By applying Strategy 1 of subsection 3 to the MBSG and the ABSG, we get:

Proposition 10. *Let s be a (finite) sequence and y be the (finite) sequence such that $y = MBSG(s)$. Let c be a randomly chosen string with the property that $MBSG(c) = y$, where the probability distribution used to choose c reflects the probability that $c = s$. The probability that, for every output bit y_k, the sequences d and c agree on the length of the tuple from which y_k arises, is $3^{-\ell}$, where ℓ denotes the length of y.*

Proof. Each candidate input string should have a tuple $(b, 0^i, 1)$ inserted for every b occurring in the output sequence, where i is chosen independently at random for each output bit, such that $i = j$ with probability 2^{-j-1}. The probability that the candidate string and the correct string agree in any one of the ℓ positions is $\sum_{i=0}^{\infty}(2^{-i-1})^2 = 1/3$. That is, the probability that the correct input sequence and a candidate c agree on the ℓ choices of length of the tuples from which the ones of y arises is $3^{-\ell} \simeq 2^{-1.585\ell}$. □

One can show that Proposition 10 also holds for the ABSG. By assuming the knowledge of no additional information on the means used to generate the input sequence, we deduce from Proposition 10 that the Hamming weight of the output sequence does not play a part in the input sequence reconstruction problem.

5.2 Strategy 2: Choice of the Most Favourable Case

In the case of the MBSG (resp. ABSG) applied to the output sequence of a maximum length LFSR of size L with a public feedback polynomial, the following attack can be mounted: it consists of finding a window of $L/2$ bits coming from a pair of bits $(b, 1)$ (resp. (b, b) for the ABSG), which occurs with probability $2^{-\frac{L}{2}}$. This window can give instantly the L bits of the current state of the LFSR. Thus, we can instantly check if we have found the correct values. In order to find the current state with high probability, we have to repeat this procedure $2^{\frac{L}{2}}$ times. This "attack" costs $\mathcal{O}(2^{\frac{L}{2}})$ and requires $\mathcal{O}(L2^{\frac{L}{2}})$ bits of keystream. This "attack" is slightly better than the generic attack thanks to the reduction in memory required.

6 FBDD-based Cryptanalysis

Krause [7] introduced a new type of attack against keystream generators, called the FBDD-attack (FBDD for Free Binary Decision Diagram), which is a cryptanalysis method for LFSR-based generators. A generator is said to be LFSR-based if it consists of two components, a linear bitstream generator LG wich generates for each initial state $x \in \{0, 1\}^n$ a linear bitstream $LG(x)$ using one or more parallel LFSRs, and a compression function C which transforms the internal bitstream into an output keystream $y = C(LG(x))$.

The cryptanalysis method relies on two assumptions called the *FBDD Assumption* and the *Pseudorandomness Assumption* (see [7] for details). The cost of the cryptanalysis depends on two parameters of the compression function C. The first parameter is the maximal number of output bits which C produces on internal bitstreams of length m; let γ be the best case compression ratio of C. Krause cryptanalysis applies when the following property is fulfilled: for all $m > 1$, the probability that $C(z)$ is a prefix of y for a randomly chosen and uniformly distributed $z \in \{0,1\}^m$ is the same for all keystreams y. Observe that both the ABSG and the MBSG have this property but the BSG does not have (nevertheless the Krause attack is expected still to work, and later we will try to estimate its complexity). Let us denote this probability $p_C(m)$. The second parameter, called α, depends on the probability $p_C(m)$. Indeed, the probability $p_C(m)$ is supposed to behave as $p_C(m) = 2^{-\alpha m}$, with α a constant such that $0 < \alpha \leq 1$. This result comes from the following partition rule: each internal bitstream z can be divided into consecutive elementary blocks $z = z^0 z^1 ... z^{s-1}$ such that $C(z) = y_0 y_1 ... y_{s-1}$ with $y_j = C(z^j)$ and the average length of the elementary blocks is a small constant. Then, we have $\alpha \approx -\frac{1}{m}\log(p_C(m))$ for large m.

Theorem 1. *[7] Let E be an LFSR-based keystream generator of key-length L with linear bitstream generator LG and a compression function C of information rate α and best case compression ratio γ. Let C fulfill the FBDD and the pseudorandomness assumption. Then, there is an $L^{O(1)}2^{(1-\alpha)(1+\alpha)L}$-time bounded algorithm which computes the secret initial state x from the first $\lceil \gamma\alpha^{-1}L \rceil$.*

Remark 4. The parameter α used to compute the complexity of the FBDD attack is **not** the information rate of the compression function, see appendix B.1 for details.

For both the ABSG and the MBSG, one can check that the FBDD Assumption and the Pseudorandomness Assumption are fulfilled and the value of γ is clearly 1/2. Our results on the FBDD attack are summarised in Table 1. In Appendix B we explain how these results are obtained.

Remark 5. We can see in the table that $\alpha_{MBSG} = \alpha_{ABSG}$. We deduce that the (time and space) complexity of the FBDD attack applied to both the ABSG and the MBSG is $L^{O(1)}2^{0.53L}$.

Remark 6. When the Krause attack is applied to BSG, the complexity depends (in a somewhat complex way) on the number of zeros in the current output sequence. Roughly speaking, with many zeros placed at the beginning of it, the attack will work better and one should apply the attack at such well chosen places in the output sequence. In Appendix B we show that the complexity ranges from $L^{O(1)}2^{0.33L}$ to $L^{O(1)}2^{0.62L}$. The best case cannot be obtained in practice, this would require $O(2^{\frac{2}{3}L})$ of keystream, and moreover $2^{0.33L}$ would still be worse than $2^{0.25L}$ obtained with the best attack of Section 3.2.

Table 1. Application of Krause FBDD attack to *BSG and SSG

		SSG	BSG	ABSG	MBSG
output rate		0.25	0.333	0.333	0.333
Krause rate γ		0.5	0.5	0.5	0.5
information rate		0.25	?	0.333	0.333
α		0.208	$0.238 \leq \alpha \leq 0.5$	0.306	0.306
Krause	time	$L^{O(1)}2^{0.66L}$	$L^{O(1)}2^{0.33L} < ... < L^{O(1)}2^{0.62L}$	$L^{O(1)}2^{0.53L}$	$L^{O(1)}2^{0.53L}$
Attack	memory	$L^{O(1)}2^{0.66L}$	$L^{O(1)}2^{0.33L} < ... < L^{O(1)}2^{0.62L}$	$L^{O(1)}2^{0.53L}$	$L^{O(1)}2^{0.53L}$

7 Conclusion

In this paper, we studied two bit-search based techniques derived from the bit-search generator. The three related compression techniques (BSG, MBSG and ABSG) studied in this paper have rate 1/3, and have good periodicity properties. Experiments suggest that they produce sequences with high linear complexity when given maximum length LFSR sequences as input. However, according to the cryptanalysis techniques that we have considered, the BSG seems less secure than both the MBSG and ABSG. Indeed, the main attack that we propose on the BSG has a complexity close to or slightly smaller than $\mathcal{O}(L2^{\frac{L}{4}})$ and requires $\mathcal{O}(2^{\frac{L}{4}})$ bits of keystream and the main attack that we propose on both the MBSG and the ABSG costs $\mathcal{O}(2^{\frac{L}{2}})$ and requires $\mathcal{O}(L2^{\frac{L}{2}})$ bits of keystream. It seems that the MBSG and the ABSG are attractive components that can be used for the de-synchronization of LFSR outputs in keystream generation.

References

1. F. Armknecht, M. Krause, *Algebraic Attacks on Combiners with Memory*, Advances in Cryptology – CRYPTO'03 Proceedings, LNCS **2729**, Springer-Verlag, (2003), 162–176.
2. D. Coppersmith, H. Krawczyk, Y. Mansour, *The Shrinking Generator*, Advances in Cryptology – CRYPTO'93 Proceedings, LNCS **773**, Springer-Verlag, D. R. Stinson, ed., (1993), 22–39.
3. N. Courtois, W. Meier, *Algebraic Attacks on Stream Ciphers with Linear Feedback*, Advances in Cryptology – EUROCRYPTO'03 Proceedings, LNCS **2656**, Springer-Verlag, (2003), 345–359.
4. P. Ekdahl, T. Johansson, W. Meier, *Predicting the Shrinking Generator with Fixed Connections*, Advances in Cryptology – EUROCRYPT 2003 Proceedings, LNCS **2656**, Springer-Verlag, E. Biham, ed., (2003), 330–344.
5. S. Golomb, *Shift Register Sequences*, Revised Edition, Aegean Park Press, (1982).

Appeared in Henri Gilbert and Helena Handschuh(Eds.): FSE 2005, LNCS 3557,pp. 196–214, 2005.

6. A. Gouget and H. Sibert. *The bit-search generator*, In *The State of the Art of Stream Ciphers: Workshop Record, Brugge, Belgium, October 2004*, pages 60–68, 2004.
7. M. Krause. *BDD-based Cryptanalysis of Keystream Generators*, In EUROCRYPT 2002, pp. 222-237, LNCS 2332, Springer, 2002.
8. W. Meier, O. Staffelbach, *The Self-Shrinking Generator*, Advances in Cryptology – EUROCRYPT'94 Proceedings, LNCS **950**, Springer-Verlag, A. DeSantis, ed., (1994), 205–214.
9. R. A. Rueppel, *Analysis and Design of Stream Ciphers*, Springer-Verlag, (1986).

A Proof of Proposition 9

We first prove the following lemma:

Lemma 1. *Let s be a periodic sequence with a period of the form:*

$$0^{\lambda_1}1^{\mu_1}0^{\lambda_2}1^{\mu_2}\dots0^{\lambda_p}1^{\mu_p} \quad ,$$

with $\lambda_i > 0$ and $\mu_i > 0$ for every i, and $\mu_p = 1$. Then, we have:

1. *the finite sequence $MBSG(t)$ is a period of $MBSG_P(s)$,*
2. *the length of $MBSG(t)$ is equal to $p + \displaystyle\sum_{i=1}^{p} \left\lfloor \frac{\mu_i - 1}{2} \right\rfloor$,*
3. *the number of zeros in $MBSG(t)$ is equal to $\#\{i, \mu_i = 1 \mod 2\}$,*
4. *the number of ones in $MBSG(t)$ is equal to $\displaystyle\sum_{i=1}^{p} \left\lfloor \frac{\mu_i}{2} \right\rfloor$.*

Proof. As the period ends with the pattern 01, we know that a cursor with initial state $\emptyset$ before reading the period is in state $\emptyset$ after reading this period. Therefore, a period of $MBSG_P(s)$ is $MBSG(0^{\lambda_1}1^{\mu_1}\dots0^{\lambda_p}1^{\mu_p})$.
Next, the output of a bit corresponds to reading a pattern of the form $(b, 0^k, 1)$, with $b \in \{0, 1\}$ and $k \geq 0$. In the periodic part of the output, these patterns necessarily contain a maximal sequence of 0's, so 0^k is some 0^{λ_i} if $b = 1$, otherwise $(b, 0^k)$ is some 0^{λ_i}. Therefore, one output bit corresponds to each maximal sequence 0^{λ_i}. There are p such sequences in the period. The other output bits come from patterns that do not contain 0, that is, from patterns $(1, 1)$. Now, in every maximal sequence 1^{μ_i}, the first 1 is the end of a pattern containing a maximal sequence of zeros. Therefore, there remains only $\lfloor\frac{\mu_i-1}{2}\rfloor$ complete pairs of ones in the sequence 1^{μ_i} in order to output bits from pairs of ones. Thus, $\sum_{i=1}^{p}\lfloor\frac{\mu_i-1}{2}\rfloor$ output bits correspond to pairs of ones in the period. This completes the second result.
We now turn to the number of zeros. A zero is output if and only if the corresponding pattern in the input is of the form $(0^{\lambda_i}, 1)$. Now, this pattern can correspond to an output bit if, and only if, the cursor is in state $\emptyset$ before the maximal sequence 0^{λ_i}. This is the case if, and only if, the length of the maximal sequence $1^{\mu_{i-1}}$ is odd. This gives the next result.
The number of ones comes directly from the two previous results. □

Appeared in Henri Gilbert and Helena Handschuh(Eds.): FSE 2005, LNCS 3557,pp. 196–214, 2005.

The proof of Proposition 9 is then a straightforward computation given the well-known distribution of maximal sequences in a period of the input, that appears for example in [5].

B The FBDD Attack Applied to the BSG, the ABSG and the MBSG

B.1 Comments on Krause Article

In [7], Krause denotes by $p_C(m)$ the probability that a randomly chosen and uniformly distributed $z \in \{0,1\}^m$ is compatible with a given keystream y, i.e., that $C(z)$ is a prefix of y. He considers only sequence generators such that this probability is the same for every y. Then, he defines $\alpha = -\frac{1}{m}\log(p_C(m))$ and he claims that α is the *information rate* per bit revealed by the keystream y about the first m bits of the corresponding internal bitstream z, i.e.

$$\alpha = \frac{1}{m}(H(Z^{(m)}) - H(Z^{(m)}|Y)) = \frac{1}{m}(m - \log(p_c(m)2^m)),$$

where $Z^{(m)}$ denotes a random $z \in \{0,1\}^m$ and Y a random keystream. This would hold if $H(Z^{(m)}|Y) = -\log(p_c(m)2^m)$ which is not always true, because, given an output keystream, not all compatible inputs are equally probable.

To clarify, the complexity of the Krause attack does indeed depend on α as defined, but this α is not in general equal to the information rate. We obtain a counterexample if we compare α_{ABSG} and the information rate of its compression function.

Computation of the information rate: we computed the information rate θ for the ABSG and the MBSG. Let m be the length of the internal bitstream, and let z denote a random, uniformly distributed element from $\{0,1\}^m$. The number of z such that $C(z)$ has length $i \geq 0$ is the number of patterns of the form $\overline{b_1}\ b_1^{k_1}\ \overline{b_1}\ \ \overline{b_2}\ b_2^{k_2}\ \overline{b_2}\ \ \ldots\ \ \overline{b_i}\ b_i^{k_i}\ \overline{b_i}\ \ \overline{b_{i+1}}b_{i+1}^{k_{i+1}}$ with $k_j \geq 0$ and $\sum_{j=1}^{i} k_i = m - 2i$. We have the following possible values for $w = \overline{b_{i+1}}b_{i+1}^{k_{i+1}}$:

- if w is the empty word or one bit (which can then be both 0 or 1), the pattern occurs with probability 2^{m-i},
- if w has length at least 2, then we have $w = \bar{b}b^k$ with $k > 0$, and only one case is possible for compatibility with the next output bit. The whole pattern occurs with probability 2^{m-i-1}.

Let $N(m)$ be the number of sequences $\overline{b_1}\ b_1^{k_1}\ \overline{b_1}\ \ \overline{b_2}\ b_2^{k_2}\ \overline{b_2}\ \ \ldots\ \ \overline{b_i}\ b_i^{k_i}\ \overline{b_i}$ of length m that are a prefix for a given y. We know that $N(m)$ is the number of ways of distributing $m - 2i$ bits among i places. The number of ways of distributing p bits among q places is a known combinatorial problem and can be written as $\binom{p+q-1}{p}$. Therefore $N(m) = \binom{m-i-1}{m-2i}$.

Then we have:

$$H_m(Z|Y) = \sum_{k=2}^{m-2}\sum_{i=1}^{\lfloor\frac{k}{2}\rfloor}\binom{k-i-1}{k-2i}\frac{m-i-1}{2^{m-i-1}} + 2\sum_{i=1}^{\lfloor\frac{m-1}{2}\rfloor}\binom{m-i-2}{m-2i-1}\frac{m-i}{2^{m-i}}$$
$$+ \sum_{i=1}^{\lfloor\frac{m}{2}\rfloor}\binom{m-i-1}{m-2i}\frac{m-i}{2^{m-i}} + \frac{m-1}{2^{m-1}}$$

Now we compute θ_{ABSG} with the formula above: we obtain $\lim_{m\to\infty}(\theta_{ABSG}) = \frac{1}{3}$, and for $m \geq 128$, we already have $\theta_{ABSG} \approx 0.33$.

Remark 7. We can also, in a very similar way, compute θ for MBSG:

$$H_m(Z|Y) = \textstyle\sum_{m=2}^{M-1} \sum_{i=1}^{\lfloor \frac{m}{2} \rfloor} \binom{m-i-1}{m-2i} \frac{M-i-1}{2^{M-i-1}} + \sum_{i=1}^{\lfloor \frac{M}{2} \rfloor} \binom{M-i-1}{M-2i} \frac{M-i}{2^{M-i}} + \frac{M-1}{2^{M-1}}$$

We also obtain $\lim_{m\to\infty}(\theta_{MBSG}) = \frac{1}{3}$, and $\theta_{MBSG} \approx 0.33$ for $m \geq 128$. At last, a similar computation for the SSG yields $\lim_{m\to\infty}(\theta_{SSG}) = \frac{1}{4}$.

B.2 The FBDD Attack Applied to the ABSG and the MBSG

Recall that the cost of the FBDD cryptanalysis depends on two parameters called α and γ. For both the ABSG and the MBSG, the best compression ratio γ is achieved when each keystream bit comes from a pattern of length 2 and we have $\gamma_{ABSG} = \gamma_{MBSG} = \frac{1}{2}$. We compute in this subsection the value of α_{ABSG} (resp. α_{MBSG}), that is, the number of possible sequences of internal bitstream z of length m such that $ABSG(z)$ (resp. $MBSG(z)$) is a prefix for a given y when z is a random and uniformly distributed element from $\{0,1\}^m$; for both the ABSG and the MBSG this number does not depends on the keystream y.

Let us consider the action of the ABSG on an input sequence z. A sequence z that produces m keystream bits, where $m \geq 0$, has two possible forms:

- $\overline{b_1}\, b_1^{k_1}\, \overline{b_1}\ \ \overline{b_2}\, b_2^{k_2}\, \overline{b_2}\ \ \ldots\ \ \overline{b_i}\, b_i^{k_i}\, \overline{b_i}$, where $k_j \geq 0$
- $\overline{b_1}\, b_1^{k_1}\, \overline{b_1}\ \ \overline{b_2}\, b_2^{k_2}\, \overline{b_2}\ \ \ldots\ \ \overline{b_i}\, b_i^{k_i}\, \overline{b_i}\ \ \overline{b_{i+1}} b_{i+1}^{k_{i+1}}$, where $k_j \geq 0$ and the last part $\overline{b_{i+1}} b_{i+1}^{k_{i+1}}$, that we call the *last word*, does not produce any bit.

Let y be an arbitrary keystream. Let B_m be the number of possible bitstream sequences z of the form $\overline{b_1}\, b_1^{k_1}\, \overline{b_1}\ \ \overline{b_2}\, b_2^{k_2}\, \overline{b_2}\ \ \ldots\ \ \overline{b_i}\, b_i^{k_i}\, \overline{b_i}$ of length m which are a prefix of y. This number does not depend on y. We know that $B_0 = 1, B_1 = 0, B_2 = 1, B_3 = 1, B_4 = 2 \ldots$. For every $m > 0$, we have $B_m = B_{m-2} + B_{m-3} + \cdots + B_0$. Indeed, if we fix the length of the first pattern $\overline{b_1}\, b_1^{k_1}\, \overline{b_1}$, the number of possibilities is then B_{m-k_1-2}.

Let A_m the number of all possible bitstream sequences z such that $ABSG(z)$ is a prefix of y. We have:

$$A_m = B_m + 2B_{m-1} + \sum_{j=0}^{m-2} B_j,$$

where B_m is the number of possible z of the first form, $2B_{m-1}$ is the number of possible z of the second form with $k_{i+1} = 0$ (when the last word contains only one bit, there are two possibilities for this bit), and the B_is for $i \leq m-2$ are the number of possible z with $k_{i+1} = m - i - 1$. Therefore we have:

$$\begin{aligned} A_m - A_{m-1} &= B_m + B_{m-1} - B_{m-2} \\ &= \sum_{i=0}^{m-2} B_i + \sum_{i=0}^{m-3} B_i - \sum_{i=0}^{m-4} B_i \quad = \quad B_{m-2} + 2B_{m-3} + \sum_{i=0}^{m-4} B_i \\ &= A_{m-2} \end{aligned}$$

Thus $A_0 = 0, A_1 = 2$ and for every $m > 1$, $A_m = A_{m-1} + A_{m-2}$. Solving this recursion gives:

$$A_m = \frac{2}{\sqrt{5}}\left((\frac{1+\sqrt{5}}{2})^m - (\frac{1-\sqrt{5}}{2})^m \right) \approx \frac{2}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^m$$

Finally when m is large enough, we compute $\alpha_{ABSG} = \log(\sqrt{5}-1) \approx 0.306$.

In the same way, one can show that $\alpha_{MBSG} = \alpha_{ABSG}$. We deduce that the (time and space) complexity of the FBDD attack applied to both the ABSG and the MBSG is $L^{O(1)}2^{0.53L}$. All our results for the FBDD attack are summarised in Table 1.

B.3 The FBDD Attack Applied to the BSG

We have seen in part 6 that, for the BSG, the probability that $C(z)$ is a prefix of y for a randomly chosen and uniformly distributed $z \in \{0,1\}^m$ is not the same for all keystreams y. Thus, it is not clear whether the FBDD attack is still relevant. In this part, we suppose it is, and we show that still the attack would not be as effective as other attacks presented in this paper. We have at least to take into account the fact that the probability we called $p_C(m)$ does depend on y.

From an attacker's point of view, the best case is when the keystream y is uniquely composed of $0s$. In this case, the value of α can be easily computed and we have $\alpha = -\frac{1}{m}\log(2^{\frac{m}{2}}) = \frac{1}{2}$.

The worst case occurs when the keystream is uniquely composed of $1s$. Let B_m denote the number of bitstream sequences of length m such that the keystream is 111....1. We have: $B_0 = 0$, $B_1 = 2$, $B_2 = 2$, $B_3 = 4$. Moreover, for $m \geq 3$, if the bitstream sequence starts by $b\ \overline{b}^{i}\ b$, then the number of possibilities is $2 \times B_{m-2-i}$. Otherwise, the bitstream sequence starts by a sequence of the form $b\ \overline{b}^{m-1}$ and there are two possible values for the bit b. Then, we have $B_m = 2 + 2(B_0 + ... + B_{m-3})$. Let $A_m = B_0 + ... + B_m$, then we get the relation: $A_m = A_{m-1} + 2A_{m-3} + 2$. By computing the limit of the series $1 - \frac{1}{m}\log(A_m - A_{m-1})$ with Magma, we obtain $\alpha \approx 0.238$.

Thus, in the general case, α belongs to the interval $[0.238, 0.5]$. If the attack can be extended, its complexity will range from $L^{O(1)}2^{0.33L}$ to $L^{O(1)}2^{0.62L}$. Then the attacker should start at the most interesting place in the output sequence, but in practice he has no hope to achieve the best-case complexity.

To obtain the best case, the attacker needs to find an all-zero subsequence with length $\frac{2}{3}L$, and this can hardly be achieved without disposing of $O(2^{\frac{2}{3}L})$ bits of keystream. Moreover an FBDD attack in $2^{0.33L}$ will still be worse than $2^{0.25L}$ we obtain in Section 3.2.

C Statistical Results

Period and linear complexity statistics for m-LFSRs filtered by the BSG are given in [6].

Appeared in Henri Gilbert and Helena Handschuh(Eds.): FSE 2005, LNCS 3557,pp. 196–214, 2005.

Table 2. Period statistics for *m*-LFSRs filtered by the ABSG

L	Average *short* period length	Minimal *short* period length	Maximal *short* period length	Average *long* period length	Minimal *long* period length	Maximal *long* period length
8	84.63	82	88	170.38	167	173
9	169	159	183	342	328	352
10	341.1	328	358	681.9	665	695
11	682.91	657	714	1364.09	1333	1390
12	1364.08	1330	1399	2730.92	2696	2765
13	2731.34	2658	2796	5459.66	5395	5533
14	5460.08	5344	5587	10922.92	10796	11039
15	10923.04	10776	11082	21843.96	21685	21991
16	21846.16	21619	22075	43688.84	43460	43916

Table 3. Linear complexity statistics for *m*-LFSRs filtered by the ABSG

L	Average *short* lin. compl.	Minimal *short* lin. compl.	Maximal *short* lin. compl.	Average *long* lin. compl.	Minimal *long* lin. compl.	Maximal *long* lin. compl.
8	84	81	88	169.38	166	173
9	167.71	158	182	340.79	326	352
10	340.2	327	358	680.83	661	695
11	680.95	654	710	1363.24	1332	1390
12	1363.33	1330	1399	2729.96	2696	2761
13	2729.80	2656	2793	5458.75	5391	5532
14	5459.17	5342	5587	10921.96	10796	11038
15	10921.47	10774	11076	21843.05	21684	21991
16	21845.28	21618	22075	43687.95	43460	43912

Table 4. Period and linear complexity statistics for *m*-LFSRs filtered by the MBSG

L	Period	Average LC	Minimal LC	Maximal LC
8	85	$84,25$	77	85
9	171	$170,46$	165	171
10	341	$339,92$	326	341
11	683	683	683	683
12	1365	$1362,53$	1347	1365
13	2731	2731	2731	2731
14	5461	5461	5461	5461
15	10923	10923	10923	10923
16	21845	$21844,35$	21833	21845

Appeared in Henri Gilbert and Helena Handschuh(Eds.): FSE 2005, LNCS 3557,pp. 196–214, 2005.