Some Attacks on the Bit-Search Generator

Martin Hell and Thomas Johansson

Dept. of Information Technology, Lund University,
P.O. Box 118, 221 00 Lund, Sweden
{martin, thomas}@it.lth.se

Abstract. The bit-search generator (BSG) was proposed in 2004 and can be seen as a variant of the shrinking and self-shrinking generators. It has the advantage that it works at rate 1/3 using only one LFSR and some selection logic. We present various attacks on the BSG based on the fact that the output sequence can be uniquely defined by the differential of the input sequence. By knowing only a small part of the output sequence we can reconstruct the key with complexity $O(L^3 2^{0.5L})$. This complexity can be significantly reduced in a data/time tradeoff manner to achieve a complexity of $O(L^3 2^{0.27L})$ if we have $O(2^{0.27L})$ of keystream. We also propose a distinguishing attack that can be very efficient if the feedback polynomial is not carefully chosen.

1 Introduction

Lately, we have seen many new proposals for stream ciphers. The aim of binary additive stream cipher is to produce a random looking sequence and then xor this sequence with the plaintext sequence to produce the ciphertext. There are several possible approaches when designing a stream cipher. A Linear Feedback Shift Register (LFSR) with primitive feedback polynomial generates sequences that possess many of the properties which we would expect from a random sequence. Because of this it is common to use an LFSR as a building block in a stream cipher. The problem with just using an LFSR is that any output bit of the LFSR can be written as a linear function in the initial state bits. This problem is solved by introducing some nonlinearity into the cipher. There are many ways to do this and some classical approaches include letting the output of several LFSRs or some state bits of one LFSR serve as input to a nonlinear Boolean function. Another common way to introduce nonlinearity is to, by some algorithm, decimate the LFSR output sequence in some irregular way. Two well known keystream generators based on this principle are the shrinking generator [1] and the self-shrinking generator [16]. Another generator related to these is the alternating step generator [11]. This generator uses one LFSR to decide the clocking of two other LFSRs.

The bit-search generator (BSG) is a keystream generator intended to be used as a stream cipher. It was introduced in 2004 by Gouget and Sibert [10] and the construction is similar to the generators mentioned above. The output of the BSG is produced by a simple algorithm, taking a pseudorandom sequence as input.

In this paper we investigate some possible attacks on the bit-search generator. Throughout the paper we assume that the pseudorandom sequence is generated by a maximum length LFSR and that the (primitive) feedback polynomial is known to the attacker.

We give an alternative description of the BSG based on the differential of the input sequence and then we describe a simple but efficient algorithm to reconstruct the differential sequence with knowledge of only a few keystream bits. By reconstructing the differential sequence we can reconstruct the original input sequence and also the key. This attack works regardless of the form of the feedback polynomial and has complexity $O(L^3 2^{0.5L})$. If we know more keystream bits we show that the complexity will be significantly decreased. More specifically, with $O(2^{0.27L})$ bits of keystream we can mount the attack with time complexity $O(L^3 2^{0.27L})$ according to our simulations. Moreover, we describe the basis for a distinguishing attack on the BSG. This attack can be made very efficient if the feedback polynomial is of low weight or if it is possible to find a low degree multiple of the feedback polynomial with low weight.

The outline of the paper is the following. In Section 2 we describe the BSG and we compare the construction with similar generators. Then, in Section 3 we present an attack that reconstructs the input sequence to the BSG algorithm. By doing this we can recover the initial state of the LFSR. Section 4 gives the framework for a possible distinguishing attack and in Section 5 we summarize some previous attacks on the shrinking, self-shrinking and the alternating step generators. We also compare these attacks with the attacks on the BSG shown in this paper. In Section 6 we give our conclusions.

2 Description of the Bit-Search Generator

In this section we describe the bit-search generator in two different but equivalent ways. First we give the original description that uses a sequence s as input, as presented in [10]. Then we give an alternative description that uses the differential sequence d of s as input. We also compare the construction to similar keystream generators.

The principle of the BSG is very simple. It consists only of an LFSR and some small selection logic, see Fig. 1. Consider a sequence $s = (s_0, s_1, s_2 \ldots)$ generated by the LFSR. The output sequence $z = (z_0, z_1, z_2 \ldots)$ is constructed from s by first letting $b = s_0$ be the first bit to search for. If the search ends immediately, i.e. $s_1 = b = s_0$ we output 0, otherwise we continue to search the sequence s until the bit we search for is found. When the correct bit is found we output 1 and we let the following bit be the next to search for. An output bit is produced after 2 input bits with probability 1/2, after 3 input bits with probability 1/4 etc. In general, an output bit is produced after $i + 1$ input bits with probability 2^{-i} so the average number of input bits needed to produce one output bit is $\sum_{i=0}^{\infty}(i + 1) \cdot 2^{-i} = 3$. This shows that the rate of the BSG is asymptotically 1/3.

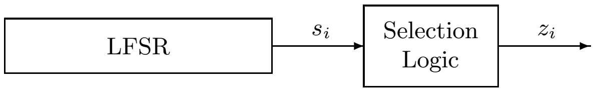

Fig. 1. Block model of the bit-search generator.

To motivate why this generator is interesting we compare it to some other well known generators based on the idea of only using LFSRs and some selection logic. We base the comparison on the number of LFSRs used and the rate of the cipher. As we can see in Table 1 the BSG has lower rate than the alternating step generator and the shrinking generator but it uses only one LFSR. The self-shrinking generator has also only one LFSR but it has lower rate.

Table 1. Comparison between the BSG and some well known generators.

Generator	Number of LFSRs needed	Rate
Alternating Step	3	1
Shrinking	2	1/2
Self-Shrinking	1	1/4
BSG	1	1/3

We now consider the differential sequence d of s. The differential sequence is defined as $d_i = s_i \oplus s_{i+1}$. If the sequence s is generated by an LFSR it is well known, see e.g. [14], that the differential sequence can be generated by the same LFSR. The two sequences differ only by some shift. When reconstructing s from d we need to guess the first bit in s, then the remaining bits are uniquely determined from d.

The output of the BSG can be uniquely described by knowledge of the differential sequence. Hence, if we can reconstruct the differential sequence we can predict the future outputs uniquely and we can also recover the key used to initialize the LFSR. The BSG operates on the differential sequence in the following way. If $d_i = 1$ we know that $s_i \neq s_{i+1}$ so we will output 1. Then we search the sequence d until we find the next $d_j = 1$. If instead $d_i = 0$ we know that we have two consecutive bits which are the same, hence we output 0. Now we know we have found the bit we search for in the original BSG and we skip the next bit since it does not matter which value it has. It is clear that the output of the BSG can be generated from either the original LFSR sequence or from the differential sequence. The following is an example of a sequence s and the corresponding differential sequence d. Applying the algorithms, we can see that they produce the same output.

$$\begin{aligned} s &= 010100100111011101011010\ldots \Rightarrow z = 110010101\ldots \\ d &= 11110110100110011110111\ldots \;\; \Rightarrow z = 110010101\ldots \end{aligned}$$

A summary of the two algorithms can be found in Fig. 2.

```
Output generated from s

i = –1;  j = –1;
while (1)
    i++;  j++;
    b = s[i];
    i++;
    if (s[i] == b)  z[j] = 0;
    else  z[j] = 1;
    while (s[i] != b)  i++;
```

```
Output generated from d

i = 0;  j = 0;
while (1)
    z[j] = d[i];
    if (d[i] == 1)
        i++;
        while (d[i] == 0) i++;
    i += 2;
    j++;
```

Fig. 2. The original BSG algorithm and an equivalent algorithm using the differential sequence d of s as input.

3 Reconstructing the Input Sequence

In this section we will describe a known plaintext attack that tries to reconstruct the differential sequence from the output sequence. In our attack we assume that we have an LFSR generating the pseudorandom sequence and that the feedback polynomial of the LFSR is known to the attacker. If we have an LFSR of length L we need to guess L bits to be able to find a candidate initial state of the LFSR. Each bit can be written as a linear function of the initial state bits and by clocking the LFSR with a candidate initial state we can see if the candidate output equals the given output.

It follows from the algorithm given in Fig. 2 that $z_i = 0$ corresponds to a 0 followed by an unknown value in the differential sequence. It is also clear that $z_i = 1$ corresponds to a 1 followed by $j \geq 0$ 0s followed by a single 1 and an unknown value. In short,

$$\begin{array}{lcl} z_i = 0 & \Rightarrow & (0, -) \\ z_i = 1 & \Rightarrow & (1, 0^j, 1, -) \end{array}$$

The probability of having j zeros is 2^{-j-1}, i.e. $z_i = 1$ corresponds to (1,1,–) with probability 1/2, (1,0,1,–) with probability 1/4, (1,0,0,1,–) with probability 1/8 etc. The expected number of inserted zeros is $\sum_{i=0}^{\infty} i \cdot 2^{-i-1} = 1$.

In the following we will denote by a the number of ones that we observe in an output sequence, b is the number of zeros in the output sequence and k is the number of zeros that are inserted in the candidate differential sequence, stemming from a set of a ones in the output sequence.

Now, assume that we have a set of a ones. There is one way to insert a total of $k = 0$ zeros and this happens with probability 2^{-a}. The number of ways to insert a total of $k = 1$ zero is $\binom{a}{1}$ and each has a probability of $2^{-a+1} \cdot 2^{-2} = 2^{-a-1}$. The number of ways to insert k zeros into a set consisting of a ones is a well known combinatorial problem and can be written as $\binom{a-1+k}{k}$, Hence, the probability of having a total of k zeros inserted will be

$$\binom{a-1+k}{k} 2^{-a-k}.$$

We construct a simple search algorithm based on these observations. The easiest way to find the correct differential sequence is to just guess the number of inserted zeros.

When we try to insert k zeros we need to look at an output sequence that satisfies $2a + b + k = L$. This is clear since every one in the output will give us two known bits and every zero will give us one known bit in the differential sequence. If we insert k extra zeros we will have a total of L bits which is enough to find the initial state. This leads us immediately to the algorithm in Fig. 3.

We start by just picking a part z' of the output sequence such that the length of z' satisfies $2a + b = L$. Then we insert $k = 0$ zeros. If this candidate is not the correct differential sequence, we delete the last bit in z'. If a 0 is deleted we try $k = 1$ next time since $b \leftarrow b - 1$ and we still require $2a + b + k = L$ to hold. For the same reason, if a 1 is deleted we try $k = 2$ next time. Every time $k \leftarrow k + 2$ we will miss some possible combinations and, hence, not the full space will be searched.

```
Search algorithm

Pick a part $z'$ of $z$ s.t. 2a+b=L;
k=0;
while (k <= $k_{max}$)
    Try all ways to insert k zeros in $z'$;
    Delete last bit in $z'$;
    if (Deleted bit == 0) k = k + 1;
    else  k = k + 2;
```

Fig. 3. The algorithm used to find the correct differential sequence.

3.1 Analysis of the Algorithm

The complexity of the algorithm and the probability of success will depend on two factors. First, the ratio between the number of zeros and the number of ones in the sequence. If we have found a z' which has many more zeros than ones, the complexity will be lower. This will also give us a higher success probability since we will delete a 0 more often than we will delete a 1. The second factor is the maximum number of zeros we will try to insert into the sequence before we give up. This is the value k_{max} in the algorithm in Fig 3. Choosing a high value for k_{max} will increase the success probability but it will also increase the complexity.

We consider the case when we choose a sequence z' at random. We expect the number of zeros in the sequence to be equal to the number of ones. We also expect that the deleted bit is 1 every second time. Moreover, when z' is of odd length, we consider the pessimistic case when $a = b + 1$. We have the following equations

$$\left.\begin{array}{r} 2a + b + k = L \\ a = b \end{array}\right\} \Rightarrow a = \left\lceil \frac{L-k}{3} \right\rceil$$

The probability of success will be

$$\sum_{k=0}^{k_{max}} \binom{\lceil \frac{L-k}{3} \rceil - 1 + k}{\lceil \frac{L-k}{3} \rceil} 2^{-\lceil \frac{L-k}{3} \rceil - k}$$

and we have a total complexity of

$$\sum_{k=0}^{k_{max}} \binom{\lceil \frac{L-k}{3} \rceil - 1 + k}{\lceil \frac{L-k}{3} \rceil}.$$

Similar equations can easily be found also if $a \neq b$. We choose k_{max} as the smallest integer such that the probability of success is > 0.5. Focusing on the expected case when $a = b$, we summarize the complexity of an attack in Table 2 with respect to the length of the LFSR (keylength). It is clear that the complexity of the attack is very close to $2^{0.5L}$ tests for all cases.

Table 2. The attack complexity when the number of zeros equals the number of ones in z'.

Keylength	k_{max}	Complexity
64	19	$2^{31.74}$
96	27	$2^{47.50}$
128	36	$2^{63.96}$
160	44	$2^{79.82}$
192	52	$2^{95.71}$
224	61	$2^{112.37}$
256	69	$2^{128.29}$

We can approximate the least number of plaintext bits needed in the expected case. With $L/3$ ones and $L/3$ zeros we will have knowledge of at least $2 \cdot L/3 +$

$L/3 = L$ bits in the input sequence. With about $2^{0.5L}$ different sequences to test we need to compare the candidate sequence with an extra $0.5L$ bits to see if the candidate is correct. Hence, by knowing about $2L/3 + L/2 = 7L/6$ bits of the keystream sequence we can reconstruct the input sequence with a complexity of $O(2^{0.5L})$ tests. In the 128 bit case we need approximately 150 bits of keystream.

3.2 A Data/Time Tradeoff

As mentioned in the previous section, it is clear that the complexity of the attack depends on the number of ones that we observe in the keystream. With a large amount of keystream we can find sequences with few ones and, hence, the attack complexity is decreased. This provides a data/time tradeoff in the attack. Assume that we want to find a part z' of z that contains at most a ones and at least b zeros, where $b > a$. Looking at a random sequence of $a+b$ bits, the probability that we find a sequence with at most a ones is given by

$$P(\#ones \leq a) = \frac{\sum_{i=0}^{a} \binom{a+b}{i}}{2^{a+b}}$$

using the approximation that sequences are independent. The number of tries needed before a desired sequence is found is geometrically distributed with an expected value of

$$\frac{2^{a+b}}{\sum_{i=0}^{a} \binom{a+b}{i}} = \frac{2^{L-a}}{\sum_{i=0}^{a} \binom{L-a}{i}}.$$

In the equality we use $2a + b = L$. Table 3 demonstrates this data/time tradeoff for the case when $L = 128$, i.e. the keylength is 128 bits. Simulations show that the time complexity and the amount of keystream needed intersect at around $2^{0.27L}$ for all L between 64 and 1024 bits.

Table 3. The data/time tradeoff based on the number of ones and zeros in z' using a 128 bit key.

Number of zeros (b) and ones (a) in z'	k_{max}	Complexity	Keystream
b = 2a	29	$2^{51.09}$	$2^{10.46}$
b = 3a	24	$2^{42.21}$	$2^{21.59}$
b = 4a	21	$2^{36.31}$	$2^{31.32}$
b = 5a	19	$2^{32.14}$	$2^{39.75}$
b = 6a	17	$2^{28.46}$	$2^{48.70}$

The complexities in Table 2 and Table 3 are given as the number of tests. To test if a candidate sequence is correct, a constant time is needed. This time can be divided into two parts. First we need to find the initial state of the LFSR by solving a system of L unknowns and L equations. This system can be solved in time L^{ω}. In theory $\omega \leq 2.376$, see [2], but the constant factor in this algorithm is expected to be very big. The fastest practical algorithm is Strassen's algorithm [19], which requires about $7 \cdot L^{\log_2 7}$ operations. For simplicity we write the time complexity for this step as L^3. We also need to clock the LFSR a sufficient number of times to compare our candidate output sequence with the observed output sequence. This second constant would also be needed in an exhaustive key search. Thus, the total time complexity for our key recovery attack is $O(L^3 2^{0.5L})$ knowing only $7L/6$ bits of the keystream. With the data/time tradeoff the complexity of the attack is $O(L^3 2^{0.27L})$ if we know $O(2^{0.27L})$ bits of the keystream. Note that these complexities are not formally derived but simulations show that they are valid (at least) up to keylengths of 1024 bits. The memory complexity of the attack is limited to the memory needed to solve the system of linear equations.

4 Distinguishing Attack

In this section we describe a possible distinguishing attack on the BSG. A distinguishing attack does not try to recover the key or any part of the input sequence. Instead, the aim is to distinguish the keystream from a purely random sequence. In the attack we assume that we have found a multiple of the feedback polynomial that is of weight w and degree h. Any multiple of a feedback polynomial will produce the same output sequence as the original polynomial. The well known fast correlation attack, see [15], depends on the existence of low weight multiples of modest degree of the LFSR feedback polynomial. Due to the importance of finding low weight multiples this subject has been studied in several papers, see [6, 20]. In [6], Golić estimates that the critical degree when polynomial multiples of weight w start to appear is $(w-1)!^{1/(w-1)} 2^{L/(w-1)}$, where L is the degree of the polynomial. Hence, a feedback polynomial of degree L is expected to have a multiple of weight w that is of degree approximately $2^{\frac{L}{w-1}}$. Now, assume that we have found a multiple of weight w that is of degree h.

The linear recurrence of the LFSR can be written as

$$0 = d_i + d_{i+\tau_1} + d_{i+\tau_2} + \ldots + d_{i+\tau_{w-1}} \tag{1}$$

where $\tau_{w-1} = h$ and $\tau_j < \tau_k,\ j < k$. A zero in the output sequence z corresponds to a zero in the differential sequence and a one in the output corresponds to a one in the differential sequence. Since the BSG has rate $1/3$ we can consider the following sums of symbols from the output sequence

$$B_i = z_i + z_{i+\frac{\tau_1}{3}} + z_{i+\frac{\tau_2}{3}} + \ldots + z_{i+\frac{\tau_{w-1}}{3}}. \tag{2}$$

We know that $B_i = 0$ if we have the correct synchronization ($d_{i+\tau_1}$ appears as $z_{i+\frac{\tau_1}{3}}$, $d_{i+\tau_2}$ appears as $z_{i+\frac{\tau_2}{3}}$ etc.) in the positions. We give an approximate value of the probability that we have synchronization in one position. With a multiple of low weight and high degree h the distance between z_i and any $z_{i+\frac{\tau_j}{3}}$ is in the order of h. Using the central limit theorem we say that the total number of inserted zeros after h outputs is normally distributed with standard deviation $\sigma \cdot \sqrt{h}$, where σ is the standard deviation for the number of inserted zeros after one output. Now, we approximate the probability that we have the correct synchronization as $h^{-\frac{1}{2}}$.

Hence, the probability that $z_{i+\frac{\tau_j}{3}}$, $1 \leq j \leq w-1$ is synchronized with z_i is approximately $h^{-\frac{1}{2}}$. The probability that all $w-1$ positions are synchronized, denoted $P(sync)$, is

$$P(sync) = (h^{-\frac{1}{2}})^{w-1} = h^{-\frac{w-1}{2}}$$

and the probability that $B_i = 0$ can be calculated as

$$\begin{aligned} P(B_i = 0) &= P(B_i = 0 \mid sync) \cdot P(sync) \\ &\quad + P(B_i = 0 \mid no\ sync) \cdot P(no\ sync) \\ &= 1 \cdot h^{-\frac{w-1}{2}} + 1/2 \cdot (1 - h^{-\frac{w-1}{2}}) \\ &= 1/2 + 1/2 \cdot h^{-\frac{w-1}{2}}. \end{aligned} \tag{3}$$

With a bias of $h^{-\frac{w-1}{2}}$ we will need about h^{w-1} samples of the output sequence to distinguish it from random. The complexity of the distinguishing attack depends on the degree of the multiple and if the degree is the expected degree, $h = 2^{\frac{L}{w-1}}$, our distinguisher needs about 2^L samples. However, if the feedback polynomial is not carefully chosen and we instead can find a multiple of low weight that is of much lower degree than expected, then the attack can be very efficient. This distinguisher can be improved in several ways. One way is to consider blocks of bits instead of individual bits.

We can consider a feedback polynomial with $h << 2^{\frac{L}{w-1}}$ as being a weak polynomial and ciphers using a weak polynomial can be efficiently attacked. Another class of weak feedback polynomials and an attack on these can be found in [5]. One can do a similar attack on the bit-search generator.

The values in the previous attack are approximated but for large h they are quite accurate. In the case where the feedback polynomial itself is of low weight, the values are not very accurate. We now describe how this attack can be mounted if the LFSR uses a feedback polynomial of some low weight w. Equation (1) will always hold for the differential sequence. To find the optimum guess for $z_{i+\frac{\tau_j}{3}}$, $1 \leq j \leq w-1$ in (2) we use the generating function for the probability of the number of clockings after λ outputs. Recall that the BSG will produce a keystream bit after two clockings with probability 1/2, after 3 clockings with probability 1/4 etc. The generating function can be written as

$$\left(\sum_{n=0}^{\infty} \frac{1}{2^n} z^{n+1} \right)^{\lambda}. \tag{4}$$

The coefficient of z^n is the probability that the LFSR has been clocked n times when the BSG has generated λ keystream bits.

By choosing the λ_j for which the coefficient of z^{τ_j} is highest we can determine which guess will give us the best probability of synchronization and we will also get the exact probability of a correct guess. We denote the probability that we guess λ_j correctly by p_{λ_j}. If $p_{\lambda_j},\ 1 \leq j \leq w-1$ are independent the probability that $B_i = 0$ can be written, similarly to (3), as

$$\begin{aligned} P(B_i = 0) &= 1 \cdot \textstyle\prod_{j=1}^{w-1} p_{\lambda_j} + 1/2 \cdot (1 - \prod_{j=1}^{w-1} p_{\lambda_j}) \\ &= 1/2 + 1/2 \cdot \textstyle\prod_{j=1}^{w-1} p_{\lambda_j}. \end{aligned}$$

With a bias of $\prod_{j=1}^{w-1} p_{\lambda_j}$ we need about

$$\frac{1}{\prod_{j=1}^{w-1} p_{\lambda_j}^2}$$

samples for a successful distinguishing attack. We end this section with a small numerical example showing the performance of this distinguisher on a low weight feedback polynomial.

Example 1. Consider the weight 5 primitive feedback polynomial $1+x^{29}+x^{66}+x^{95}+x^{128}$. Write the linear recurrence in the differential sequence as

$$0 = d_i + d_{i+29} + d_{i+66} + d_{i+95} + d_{i+128}.$$

Using (4) we find that the highest coefficient for z^{29}, z^{66}, z^{95} and z^{128} is achieved when we have $\lambda_1 = 10$, $\lambda_2 = 22$, $\lambda_3 = 32$ and $\lambda_4 = 43$ respectively. The best possible approximation of (2) is then $B_i = z_i + z_{i+10} + z_{i+22} + z_{i+32} + z_{i+43}$. The probability that each of these terms are synchronized with z_i is the coefficient for each term in (4), i.e.,

$$p_{\lambda_1} = 2^{-3.43}, \quad p_{\lambda_2} = 2^{-4.06}, \quad p_{\lambda_3} = 2^{-4.31}, \quad p_{\lambda_4} = 2^{-4.53}.$$

This gives us a total bias of $\prod_{j=1}^{w-1} p_{\lambda_j} = 2^{-16.33}$ and, hence, our distinguisher needs approximately $2^{32.66}$ bits to succeed.

This shows that low weight feedback polynomials can be easily and efficiently attacked. Note that the attack described above can be further improved, using slightly more advanced techniques.

5 Comparison with the Alternating Step, Shrinking and the Self-Shrinking Generator

The shrinking generator, the self-shrinking generator and the alternating step generator are similar to the BSG in that they only contain one or more LFSRs and some selection logic. There is no Boolean function used as a nonlinear combiner or as a nonlinear filter. In section 2 we compared the number of LFSRs used in and the rate of these generators. Here we summarize a small selection of the attacks proposed for the 3 well known generators and we compare them to our attacks on the BSG.

The alternating step generator is the oldest generator and based on 3 LFSRs in such a way that L_3 controls the clocking of L_1 and L_2. In the original paper [11] a divide-and-conquer attack with complexity $O(2^{L_3+2\log_2 \min(L_1,L_2)})$ was shown. 1997, in [8], Golić and Menicocci showed a correlation attack with complexity $O(2^{L_1+L_2+2\log_2(L_1+L_2)})$ and the year after [9] they improved this attack significantly to $O(2^{\max(L_1,L_2)+2\log_2 \max(L_1,L_2)})$.

The shrinking generator uses two LFSRs, denoted A and S. The sequence generated by S is used to select bits in the A-sequence. These selected bits are the output bits. In the original paper [1], an attack with known feedback polynomials was proposed that has complexity $O(2^{L_S} \cdot L_A^3)$. In 1998, Simpson, Golić and Dawson [18] presented a correlation attack that can recover the initial state of A with complexity $O(2^{L_A} \cdot L_A^2)$ using about $20 \cdot L_A$ bits. In 1998, Johansson [12] gave another correlation attack that is based on finding weak sequences. The complexity of this attack is better than previous attacks but it is still exponential in $|A|$. Distinguishing attacks on the shrinking generator have also been presented in [3, 7].

The BSG is probably most related to the self-shrinking generator since they both consist of only one LFSR. Because of this the attacks on the self-shrinking generator are easy to compare to the attacks on the BSG. Several key recovery attacks have been proposed for the self-shrinking generator. In the original paper [16] a key recovery attack was proposed that has average complexity $O(2^{0.75L})$. In 1996, Mihaljevic [17] presented an attack that has a complexity that varies between $O(2^{0.5L})$ and $O(2^{0.75L})$ but the required length of the keystream varies between $O(2^{0.5L})$ and $O(2^{0.25L})$ respectively. In 2001, Zenner, Krause and Lucks [21] described an attack that uses a search tree. This attack needs very few keystream bits and has complexity $O(2^{0.69L})$. The attack was later improved by Krause [13] to a complexity of $O(2^{0.66L})$. The problem with these two attacks is that they require a large amount of memory. In 2003, Ekdahl, Johansson and Meier [4] presented an attack that is much more efficient than the previous attacks if the polynomial is of a certain form.

Our attack that reconstructs the input sequence of the bit-search generator is equivalent to a key recovery attack. We can reconstruct the initial state of the LFSR that produces the differential sequence d. From this sequence we can reconstruct the original sequence. We propose an attack with complexity $O(L^3 2^{0.5L})$ that uses very few keystream bits. Knowing more keystream bits will reduce the complexity significantly. Using a data/time tradeoff we show that we can mount the attack using $O(2^{0.27L})$ keystream bits with a time complexity of $O(L^3 2^{0.27L})$. Finally, we suggest a distinguishing attack that can be very efficient if the feedback polynomial is not carefully chosen. It has the complexity $O(h^{w-1})$ for a degree h multiple of weight w. The framework of this distinguishing attack can be used also to attack the other generators. Hence, for all the generators considered here it is important to choose a feedback polynomial that has no low weight multiples of degree much lower than expected.

Finally, we would like to mention that the BSG, as well as the other generators in this section, can be vulnerable to various side channel attacks. Though, we have not pursued any work in this direction.

6 Conclusion

The bit-search generator, recently proposed by Gouget and Sibert has been considered and an equivalent description based on the differential of the input sequence has been given. We propose an efficient attack that recovers the differential sequence, and hence, the key. The construction as well as the security of the generator has been compared to similar generators. The self-shrinking generator is very similar to the BSG and we find that the key recovery attacks presented here are more efficient than any known key recovery attack on the self-shrinking generator. The basis for a distinguishing attack is also described and we show that if the feedback polynomial is not carefully chosen, the BSG may be prone to efficient distinguishing attacks.

References

1. D. Coppersmith, H. Krawczyk and Y. Mansour. The Shrinking Generator. In D. Stinson, editor, *Advances in Cryptology–CRYPTO'93*, pages 22–39. Springer-Verlag, 1993. Lecture Notes in Computer Science Volume 773.
2. D. Coppersmith and S. Winograd. Matrix Multiplication via Arithmetic Progressions, *J. Symbolic Computation (1990)*, 9, pp. 251–280.
3. P. Ekdahl, W. Meier and T. Johansson. Predicting the Shrinking Generator with Fixed Connections. In E. Biham, editor, *Advances in Cryptology–EUROCRYPT 2003*, volume 2656 of Lecture Notes in Computer Science, pages 330–344. Springer-Verlag, 2003.
4. P. Ekdahl, T. Johansson and W. Meier. A Note on the Self-Shrinking Generator. In *Proceedings of International Symposium on Information Theory*, page 166. IEEE, 2003.
5. H. Englund, M. Hell and T. Johansson. Correlation Attack Using a New Class of Weak Feedback Polynomials. In, B. Roy and W. Meier, editors, *Fast Software Encryption 2004*, volume 3017 of Lecture Notes in Computer Science, pages 127–142. Springer-Verlag 2004.
6. J. D. Golić. Computation of Low-Weight Parity Check Polynomials. *Electronic Letters* 32(21):1981-1982, October 1996.
7. J. D. Golić. Correlation Analysis of the Shrinking Generator. In J. Kilian, editor, *Advances in Cryptology–CRYPTO 2001*, volume 2139 of Lecture Notes in Computer Science, pages 440–457. Springer-Verlag, 2001.
8. J. D. Golić and R. Menicocci. Edit Distance Correlation Attack on the Alternating Step Generator. In, B. S. Kaliski, editor, *Advances in Cryptology–CRYPTO'97*, Volume 1294 of Lecture Notes in Computer Science, pages 499–512. Springer-Verlag 1997.

9. J. D. Golić and R. Menicocci. Edit Probability Correlation Attack on the Alternating Step Generator. In C. Ding, T. Helleseth and H. Niederreiter, editors, *Sequences and their Applications–SETA'98.* Discrete Mathematics and Theoretical Computer Science, pages 213–227. Springer-Verlag 1999.
10. A. Gouget, 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.
11. C. G. Günther. Alternating Step Generators Controlled by de Bruijn Sequences. In D. Chaum and W. L. Price, editors, *Advances in Cryptology–EUROCRYPT'87*, pages 5–14. Springer-Verlag, 1988. Lecture Notes in Computer Science Volume 304.
12. T. Johansson. Reduced Complexity Correlation Attacks on Two Clock-Controlled Generators. In K. Otha and D. Pei, editors, *Advances in Cryptology–ASIACRYPT'98*, volume 1541 of Lecture Notes in Computer Science, pages 342–357. Springer-Verlag, 1998.
13. M. Krause. BDD-Based Cryptanalysis of Keystream Generators. In L. R. Knudsen, editor, *Advances in Cryptology–EUROCRYPT 2002*, Volume 304 of Lecture Notes in Computer Science, pages 222–237. Springer-Verlag, 2002.
14. R. J. McEliece. Finite Fields for Computer Scientists and Engineers. Kluwer Academic Publishers. 1987.
15. W. Meier and O. Staffelbach. Fast Correlation Attacks on Stream Ciphers. *Advances in Cryptology-EUROCRYPT'88*, volume 330 of *Journal of Cryptology*, pages 310-314. Springer-Verlag, 1988.
16. W. Meier and O. Staffelbach. The Self-Shrinking Generator. In A. De Santis, editor, *Advances in Cryptology–EUROCRYPT'94*, pages 205–214. Springer-Verlag, 1995. Lecture Notes in Computer Science Volume 950.
17. M. Mihaljevic. A Faster Cryptanalysis of the Self-Shrinking Generator. In J. Pieprzyk and J. Seberry, editors, *First Australasian Conference on Information Security and Privacy ACISP'96*, volume 1172 of Lecture Notes in Computer Science, pages 182–189, Springer-Verlag, 1996.
18. L. Simpson, J. D. Golić and E. Dawson. A Probabilistic Correlation Attack on the Shrinking Generator. In C. Boyd and E. Dawson, editors, *Information security and privacy '98*, volume 1438 of Lecture Notes in Computer Science, pages 147–158, Springer-Verlag, 1998.
19. V. Strassen. Gaussian Elimination is Not Optimal, *Numerische Mathematik*, vol. 13, pages 354-356, 1969.
20. D. Wagner. A Generalized Birthday Problem. In M. Yung, editor, *Advances in Cryptology–CRYPT0 2002*, volume 2442 of Lecture Notes in Computer Science, pages 288–303, Springer-Verlag, 2002.
21. E. Zenner, M. Krause and S. Lucks. Improved Cryptanalysis of the Self-Shrinking Generator. In *ACIPS'2001*, Volume 2119 of Lecture Notes in Computer Science, pages 21–35. Springer-Verlag, 2001.

[0] The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability