Guess-and-determine Algebraic Attack on the Self-Shrinking Generator*

Blandine Debraize[1,2] and Louis Goubin[2]

[1] Gemalto, Meudon, France
blandine.debraize@gemalto.com
[2] Versailles Saint-Quentin-en-Yvelines University, France
Louis.Goubin@prism.uvsq.fr

Abstract. The self-shrinking Generator (SSG) was proposed by Meier and Staffelbach at Eurocrypt'94. Two similar guess-and-determine attacks were independently proposed by Hell-Johansson and Zhang-Feng in 2006, and give the best time/data tradeoff on this cipher so far. These attacks do not depend on the Hamming weight of the feedback polynomial (defining the LFSR in SSG).
In this paper we propose a new attack strategy against SSG, when the Hamming weight is at most 5. For this case we obtain a better tradeoff than all previously known attacks (including Hell-Johansson and Zhang-Feng). Our main idea consists in guessing some information about the internal bitstream of the SSG, and expressing this information by a system of polynomial equations in the still unknown key bits. From a practical point of view, we show that using a SAT solver, such as MiniSAT, is the best way of solving this polynomial system.
Since Meier and Staffelbach original paper, avoiding low Hamming weight feedback polynomials has been a widely believed principle. However this rule did not materialize in previous recent attacks. With the new attacks described in this paper, we show explicitly that this principle remains true.

Keywords: stream cipher, guess-and-determine attacks, multivariate quadratic equations, SAT solver, self-shrinking generator, algebraic cryptanalysis.

1 Introduction

The self-shrinking generator (**SSG**) was proposed by W. Meier and O. Staffelbach at Eurocrypt'94 in [12]. It is a variant of the original Shrinking Generator proposed by Coppersmith, Krawczyk and Mansour in [4, 10]. In their paper, they proposed an attack of time complexity $\mathcal{O}(2^{0.75n})$, and

* This work has been partially supported by the French Agence Nationale de la Recherche (ANR) under the Odyssee project

$\mathcal{O}(2^{0.69n})$ when the Hamming weight of the feedback polynomial is 3. In [13], Mihaljević proposed a cryptanalysis with minimal time complexity $\mathcal{O}(2^{0.5n})$, with data complexity $\mathcal{O}(n2^{0.5n})$. The amount of keystream is not realistic for large values of the key size n. An attack on SSG requiring very few keystream data ($2.41n$) is the BBD cryptanalysis proposed in [9] with time complexity $n^{\mathcal{O}(1)}2^{0.656n}$ and equivalent memory complexity. The best tradeoff between time, memory and data complexity today is the Hell and Johansson guess-and-determine cryptanalysis of [8]. A very similar attack has been independently proposed by Zhang and Feng in [14]. For instance the time complexity of this latter attack varies from $\mathcal{O}(2^{0.5n})$ to $\mathcal{O}(n^3 2^{0.666n})$ and data complexity ranges from $\mathcal{O}(n2^{0.5n})$ to $\mathcal{O}(n)$ accordingly. For example with a reasonable amount of keystream of $\mathcal{O}(2^{0.161n})$, it is possible with this attack to recover the key in time $\mathcal{O}(n^3 2^{0.556n})$. The complexity of this attack is independent from the Hamming weight of the feedback polynomial.

In this paper we show that a low Hamming weight for the feedback polynomial defining the LFSR makes the self-shrinking generator even more vulnerable against guess-and-determine attacks. To show this we propose a new type of guess-and-determine attack. We guess some information and then write a system of polynomial equations over GF(2) that we solve by using the SAT solver algorithm MiniSAT. We describe a large family of attacks. Thus as the Hell-Johansson and Zhang-Feng attacks, we can handle with different conditions of attack and data requirements. Our simulations show that for small Hamming weight feedback polynomial, the complexity of our time/data tradeoff is noticeably better.

In Section 2, we briefly describe SAT solvers, the design of the SSG and the principle of our attack. In Section 3 we analyse previous work on this cipher. In Section 4, we describe a special case of our new attack, and in Section 5, we generalize the principle to a family of attacks. Finally in Section 6, we look for the best time/data tradeoff cryptanalysis.

2 Preliminaries

2.1 SAT Solvers

In cryptography the use of SAT solvers to solve polynomial systems over GF(2) has been recently introduced by Bard, Courtois and Jefferson in [1, 2]. The method consists of converting the multivariate system into a conjunctive normal form satisfiability (CNF-SAT) problem, and then applying a SAT solver algorithm. It has been used in [3] to cryptanalyse the block cipher Keeloq and in [11] to analyse the reduced version Bivium of the stream cipher Trivium.

The other well-known methods to solve algebraic systems of equations over GF(2) are XL ([5]) and Gröbner bases algorithms like F4 and F5 ([6, 7]). Both are linear algebra based methods, their drawback is that they need to store big matrices during the computations and then require a huge amount of memory. Moreover it is unclear how much the sparsity of the initial system helps to reduce the running time of the solving.

SAT solvers behave in a completely different way. Most of them try to find more directly a solution to the system by recursively choosing a variable, first trying to assign it a value and then the other. The important parameters for SAT solvers are the number of clauses, the total length of all the clauses, and the number of variables.

In this paper we use the conversion from algebraic normal form to conjunctive normal form method described in [2], and the SAT solver MiniSAT also proposed in [2]. This conversion method transforms linear equations in long CNF expressions made of long clauses. That is why the method works much better if the linear expressions are short, and, more generally, if the systems are sparse.

2.2 Trade-off between Guessing and Exploiting Information

In this Section, we specifically consider the case of stream ciphers based on one Linear Feedback Shift Register (LFSR), since the self-shrinking generator belongs to this category. However, the notions defined below can be extended to stream ciphers based on several LFSRs.

Let us suppose the state of the LFSR has length n. At each clock t, the LFSR outputs a bit s_t. The bits $s_0, \cdots, s_{n-1}$ are the bits of the initial state of the LFSR. Here we consider that the initial state of the LFSR is the n-bit key of the cipher.

We call internal sequence at clock t the sequence of bits $S^t = s_0 s_1 ... s_t$. At each clock t, the compression function outputs one bit or an empty word $C(S^t)$. The compression ratio η is the average number of output bits generated by one bit of random internal sequence. For the SSG the compression ratio is $\eta = \frac{1}{4}$.

Definition 1. *The information rate (per bit), which a keystream reveals about the first m bits of the underlying internal bitstream, is denoted by $\alpha(m)$, and defined by $\alpha(m) = \frac{1}{m} I(Z^{(m)}, Y)$, where $Z^{(m)}$ denotes a random $z \in \{0,1\}^m$ and Y a random keystream.*

Then $\alpha(m)$ can be computed as:

$$\alpha(m) = \frac{1}{m} I(Z^{(m)}, Y) = \frac{1}{m} \Big(H(Z^{(m)}) - H(Z^{(m)}|Y) \Big) = 1 - \frac{1}{m} H(Z^{(m)}|Y)$$

We prove in appendix A that the information rate is constant for the self-shrinking generator and that its value is $\frac{1}{4}$.

For a stream cipher based on one LFSR with a constant information rate and a constant compression ratio, there is always a better attack than exhaustive search, by exploiting the leakage of information given by the keystream. For m keystream bits, this leakage is an amount of $\alpha m/\eta$ bits of information. The entropy of the guess to recover the m/η first internal sequence bits is then $H(Z^{(m)}|Y) = (1-\alpha)\frac{m}{\eta}$. Recovering the n key bits requires then a complexity $\mathcal{O}(2^{(1-\alpha)n})$. This attack has been described in [12]. One way to improve this attack is to decrease the amount of information we guess. In this case we cannot recover directly all the consecutive bits of the initial state of the LFSR, but only part of them. If we guess an amount of information h on the internal sequence per keystream bit, what we obtain is an amount of $h + \alpha/\eta$ per keystream bit. The ratio "guessed information"/"total information known per keystream bit" is then

$$\frac{h}{h + \frac{\alpha}{\eta}}$$

where $\frac{\alpha}{\eta}$ is a constant (here equal to 1). Therefore the smaller h gets, the smaller this ratio becomes. This means that when h decreases, the amount of "guessed information" staying the same, the obtained "total information" increases.

Decreasing the amount of information on the internal sequence we guess per keystream bit seems then to be a good strategy. It is the adopted strategy throughout this paper. The greatest issue is the following: once we have obtained enough information, how to exploit it to recover the key. This will be discussed in detail in this paper for the case of the self-shrinking generator.

2.3 Description of the Self-Shrinking Generator

The self-shrinking generator consists of one LFSR, and a shrinking component that uses a compression function C. Let $K = (K_0, \cdots, K_{n-1})$ be a secret key, and let $s^0 = K$ be the initial state of the LFSR. At each clock

$t = 0, 1, 2, \cdots$, the new state s^t is computed as $s^t = L(s^{t-1})$, with L being the multivariate linear transformation corresponding to the connection polynomial of the LFSR. Therefore $s^t = L^t(K_0, \cdots, K_{n-1})$, and every bit s_i^t of the state at time t can be written as a known linear combination of the key bits $K_0, \cdots, K_{n-1}$.

Now we define the compression function. Let f be a function defined as follows :

$$f : \{0,1\}^2 \longrightarrow \{0, 1, \varepsilon\}$$

such that $f(a, b) = b$ if $a = 1$, and $f(a, b) = \varepsilon$ (the empty word) if $a = 0$. This compression function can be extended to compress sequences of bits of arbitrary length as follows. Let $x_0\, x_1 \cdots x_{r-1}$ be a bitstream of length r generated by the LFSR. The output keystream of the SSG generator will be $C(x_0\, x_1 \cdots x_{r-1})$, which is defined as $f(x_0, x_1)\, f(x_2, x_3) \cdots f(x_{r-2}, x_{r-1})$ with the computation being done in the free monoid $\{0, 1\}^*$ (which means that we simply concatenate these strings of bits). The resulting compressed sequence $C(x_0\, x_1 \cdots x_{r-1})$ has length at most $\lceil \frac{r}{2} \rceil$. This length is hard to predict and depends on the number of pairs of consecutive bits such that $f(x_i, x_{i+1}) = \varepsilon$ (*i.e.* $x_i = 0$ and no bit is output).

3 Previous Work and Known attacks

3.1 The Meier and Staffelbach Attack

The attack described in [12] is the attack we refereed to in Section 2.2. It consists of guessing all the consecutive bits of an internal sequence s of length n that are not revealed by the keystream. As the compression ratio is $\frac{1}{4}$, the amount of unknown bits is on average $\frac{3n}{4}$. As announced in Section 2.2, the complexity of this attack is $2^{\frac{3}{4}n}$.

Two completely different attacks were proposed in 2001 [15] of complexity $\mathcal{O}(2^{0.694n})$, and in 2002 [9] of complexity $n^{\mathcal{O}(1)}2^{0.656n}$, for which we will not go into detail in this paper.

There are two ways of improving Meier and Staffelbach attack. The first one consists of reducing the amount of information we guess, as we describe in Section 3.2. The second one consists in looking for the best case through the keystream, as we briefly describe in Section 3.3.

3.2 Improvement

It is easy to improve this attack by decreasing the amount of information we guess. The known method we explain here can be found in [8]. Each bit x_i of the pseudo-random sequence corresponds to two consecutive bits 1 and x_i in the internal sequence s. Then it is possible, instead of guessing the values of all the bits of the internal sequence, to guess only the values of the subsequence s' made of the even bits of s $(x_0, x_2, \cdots, x_{2n}, \cdots)$. It is equivalent to guessing the position of the pairs $(1, x_i)$ in s. We show now that this decreases the amount of information we guess per bit. Let us suppose that $x_0 = 1$. The probability for the number of "0" to be k before the next "1" in s' is $\frac{1}{2^{k+1}}$. Consequently the entropy for this information is

$$H(L) = \sum_{j=0}^{+\infty} \frac{j+1}{2^{j+1}} = 2$$

Let us suppose we get a sequence of m bits of keystream. The entropy for guessing the values of the corresponding internal sequence s' (bits in even positions) is then $2m$. Therefore we can guess all these values with an average about 2^{2m} guesses. We have seen that the i-th bit of the keystream is equal to the odd bit following the i-th even "1" of s. Once we get the positions of the "1"s in the internal bitstream, we know the values and positions of $2m + m = 3m$ bits on average. Therefore m must be about $\frac{n}{3}$, assuming there is no redundancy in the information.

How to exploit this information? Here it is very simple, as each internal sequence bit equals a linear expression of the key bits. We have then obtained a system of linear equations. The non-redundancy of the information obtained by our guess is expressed by the consistency of this linear system.

We observe that in this attack the ratio "information guessed"/"information obtained" is $\frac{2}{3}$.

3.3 Mihaljević Attack

This attack is described in [13]. Let us consider again the subsequence s' of the internal bitstream made of the even bits. When we know that $\frac{n}{2}$ consecutive bits of s' are "1"s, we know n consecutive bits of s. The attack consists in looking for this case through the keystream. To each keystream subsequence of $\frac{n}{2}$ bits corresponds an n-bit internal bitstream sequence. If by running the stream cipher on this sequence we do not obtain right values for the keystream, we try on the following $\frac{n}{2}$ bits sequence of keystream, etc.

Of course the drawback of this attack is the huge amount of necessary keystream bits: about $\frac{n}{2} \cdot 2^{\frac{n}{2}}$. This is why [13] describes a family of attacks with time complexity varying from $\mathcal{O}(2^{\frac{n}{2}})$ (this attack) to $\mathcal{O}(2^{\frac{3n}{4}})$ (the attack of Section 3.1), and the required keystream length ranging from $2^{\frac{n}{2}}$ to $2^{\frac{n}{4}}$ accordingly.

The other tradeoff between the attack allowing the best complexity estimation and the attack described at Section 3.2 is studied in [14] and [8]. The attack strategy is the same in both papers, but in [8], an improvement is proposed when the available keystream is very short (less than $2^{0.05n}$). As our final attack will only focus on larger keystream amounts, we will only take into account the common part of [14] and [8] in this paper. We will briefly describe it in Section 6.

4 Principle of Our Attack

Our aim is to generalize the method described at Section 3.2. In this attack we guess some bit values and solve the system of linear equations by a Gaussian elimination when the system of linear equations has rank n.

To adopt a more general point of view on this attack, we can say that we exploit the information we have obtained when its amount is sufficient, *i.e.* when we have obtained n bits of information on the key (recall that the key is the initial state of the LFSR). In Section 3.2 we exploit this information by a linear algebra method. Each linear equation in the key bits represents one bit of information. Here the non-redundancy of the information obtained is guaranteed by the independence of the linear equations.

In the following, we keep this point of view. We guess some information on the internal sequence and directly compute the total amount of information we have obtained. The second step consists then in exploiting this information by completely describing it by a system of polynomial equations and solving this system with algebraic techniques.

4.1 Guessing Information

In the attack of Section 3.1, the amount of information that is guessed per keystream bit is 3. In the attack of Section 3.2, it is 2. What we want to do here is to further decrease this amount of guessed information per bit. Instead of guessing the positions of the "1"s of the subsequence s' made of the even bits of the internal sequence, such as in the attack of Section 3.2, we guess the positions of one such bit out of two.

Let us consider a sequence of keystream bits $x_i, x_{i+1}, \cdots, x_{i+k}, \cdots$. Each of these bits x_j correspond to a pair $(1, x_j)$ in the internal bitstream s. Then we guess the positions of the corresponding pairs for

$x_i, x_{i+2}, x_{i+4}, \cdots, x_{i+2k'}, \cdots$. Thus for example the precise position of the pair corresponding to x_{i+1} is unknown but ranges between the position of the pair corresponding to x_i and the position of the pair corresponding to x_{i+2}.

Let us define for this attack a "block" of internal sequence bits: each block contains two pairs beginning by 1 and the pairs beginning by "0" until the next "1" in the sequence. This means that each block begins by a "1". For example, if the internal sequence is :

$$01\ 10\ 00\ 01\ 10\ 00\ 10\ 00 \cdots,$$

the first block we find for this sequence is 10 00 01 10 00.

To know the position of one 1 out of two in s', it is enough to guess the size of consecutive blocks of s, *i.e.* to guess the number of pairs beginning by 0 in each block. The probability to have k pairs beginning by 0 in a block is the number of ways of distributing k bits among 2 places multiplied by $\frac{1}{2^{k+2}}$. For any q, the number of possibilities to distribute k bits among q places is $\binom{q-1+k}{k}$. The probability is then here $\frac{k+1}{2^{k+2}}$. The entropy of the information guessed by keystream bit is:

$$H = -\frac{1}{2} \sum_{k \geq 0} \frac{\binom{k+1}{k}}{2^{k+2}} \log\left(\frac{\binom{k+1}{k}}{2^{k+2}}\right) \approx 1.356$$

This information describes the fact that we know that the first even bit of the block is "1", and that the other even bits are all "0" but one. The total amount of information we know on this block comes from this information and from the fact that we know the values of the keystream bits corresponding to the even "1"s of the block, *i.e.* two bits of information. The average information we know about one block is then $2 \times 1.356 + 2$, and the known information per keystream bit is $1.356 + 1 = 2.356$. Thus for m bits of keystream, we get $2.356m$ bits of information if there is no redundancy. Then m must be approximately $\frac{n}{2.356}$ and the average complexity for the guessing part of the attack is $2^{\frac{1.356n}{2.356}} = 2^{0.575n}$.

4.2 Exploiting the information

The next stage of the attack consists in exploiting the information we have obtained. This information cannot be expressed only by linear GF(2) relations any more. But as we will see now, it is possible to describe it by quadratic equations. To ease the understanding, we call "subblock" all the pairs of a block but the first one. What we have to describe for each block is:

1. The fact that the first and second bits of the block are known. This can still be described by linear relationships.
2. The fact that only one pair among the pairs of the subblock begins by “1”. This information can be divided into two parts:
   - There is at most one “1” among the even bits of the subblock. This means that for each even bit of the subblock x_i, if x_j is another even bit of the subblock, we have:
   $$(x_i = 1) \Rightarrow (x_j = 0)$$
   This is equivalent to : $x_i x_j = 0$. Then this part of the information can be described by $\binom{k}{2}$ quadratic equations in the internal sequence bits.
   - There is at least one “1” among the even bits ot the subblock. This is described by a linear equation:
   $$\bigoplus_{j=1}^{k+1} x_{i_j} = 1$$
   where the x_{i_j} are all the even bits of the subblock.
3. The fact that the bit of the pair beginning by “1” in the subblock is known. This is described by the fact that for each even bit x_j of the subblock,
   $$(x_j = 1) \Rightarrow (x_{j+1} = e)$$
   where e is the corresponding keystream bit. It can be translated by $k+1$ quadratic boolean equations:
   $$x_j(x_{j+1} + e) = 0$$

As the composition of linear functions with quadratic equations is still quadratic, those equations can be written as quadratic equations in the key bits. We have then obtained a system of quadratic equations over the field GF(2), completely describing the key.

When the blocks are short, it is possible to find some other equations describing the information. It is interesting to have the most overdefined possible system of equations if programs like Gröbner basis algorithm or XL are used to solve the system. But in this paper we use SAT solver algorithms for which working on very overdefined systems is not the best strategy. That is why we do not add these additional equations in our systems.

We give here the results of our computations on these systems of equations for different sizes of LFSR state n and three different Hamming weights hw for the feedback polynomial of the LFSR:

Table 1. MiniSAT computations on quadratic systems of equations

	$hw = 5$	$hw = 6$	$hw = 7$
$n = 128$	$0.02s$	$0.03s$	$0.05s$
$n = 256$	$0.025s$	$0.046s$	$62s$
$n = 512$	$0.127s$	$> 24h$	$> 24h$
$n = 1024$	$122.25s$	$> 24h$	$> 24h$

5 Generalisation of the Attack

5.1 Guessing Information

This method can be generalized. In Section 4, we have chosen to guess the position of one even "1" in the internal sequence out of $q = 2$. Now we can choose to guess the position of one 1 out of q bits, with $q \geq 2$. This is again equivalent to guessing the length of the "blocks" made of the consecutive bits of the internal sequence containing q pairs of bits beginning by "1" and the other pairs beginning by "0" until the next even "1".

Each such block correspond to q keystream bits. The average entropy per keystream bit to guess the length of consecutive blocks is then:

$$H(q) = -\frac{1}{q} \sum_{k \geq 1} \frac{\binom{q-1+k}{k}}{2^{q+k}} \log\left(\frac{\binom{q-1+k}{k}}{2^{q+k}}\right)$$

For example, when $q = 3$, $H(q) = 1.0385$.

As explained in Section 2.2, the total amount of information we obtain per keystream bit is $1+H(q)$. If there is no redundancy, it is then necessary to guess the length of the blocks corresponding to $\frac{n}{1+H(q)}$ keystream bits and the average complexity of the guess is $2^{\frac{H(q)}{1+H(q)}n}$.

Table 2. Average complexity of the guess for various values of q

	$q = 2$	$q = 3$	$q = 4$	$q = 5$
Complexity	$2^{0.575n}$	$2^{0.509n}$	$2^{0.458n}$	$2^{0.417n}$

5.2 Solving the Polynomial System - Computational Results

As in Section 4.2, we need to completely describe the amount of information we have, by means of polynomial GF(2) equations. It is possible to describe it with equations of degree at most q, in a way very similar to the one proposed at Section 4.2. We give details in Appendix B.

Moreover, it is possible to show that for small blocks, the degree of the equations decreases. If Gröbner bases are used, it is well known that the smaller the degree is, the faster the attack is also. With SAT solvers, even if this correlation is not so clear, our computations showed that the complexity gets smaller when the degree of polynomials gets smaller. This tends to show that the shorter the blocks are, the faster the complexity of solving the system is. We will exploit this at Section 6.

We have written the systems of equations for $q = 3$ and $q = 4$ for values of n ranging from 128 to 512. We fixed the value of the Hamming weight of the feedback polynomial to 5 as greater values seem to lead to much slower attacks. We then applied our SAT solver algorithm on these systems. We give the results of the computations in table 3.

Table 3. MiniSAT computations on quadratic systems of equations for q=3 and q=4

	$n = 128$	$n = 256$	$n = 512$
$q = 3$	$2.28s$	$80s$	$2716s$
$q = 4$	$14s$	$1728s$	$> 24h$

6 Improvement of the General Attack

In the previous Sections, we have seen that the basic attack of [12] can be extended in two directions. The first one (first proposed by Mihaljević in [13]) looks for a tradeoff between time complexity and required keystream length. The second one, especially studied in this paper, looks for a tradeoff between the cost of guessing information and the cost of exploiting this information. The best attack consists then of choosing the best tradeoff in both directions at the same time.

In [14] and [8], an attack is proposed that is already a tradeoff between a similar attack as the one described in Section 3.2 and the best time complexity attack proposed in [13], when the length of keystream is maximal. The authors guess all the even bits of a sequence of the internal bitstream of length l, assuming that the rate of "1"s in these even bits is at least α (with a fixed $\alpha > \frac{1}{2}$). They choose the value l, depending on α, in order to have enough information to recover the key by a Gaussian elimination once they have guessed all the even bits of the sequence. In order to find such a sequence, they go through the keystream. The time and data complexity completely depend on the value chosen for α. For instance, the authors of [14] Zhang and Feng obtain a time complexity of $\mathcal{O}(n^3 2^{\frac{n}{1+\alpha}})$.

In Section 5, we denoted by q the number of even "1"s in a block, and considered guessing the position of one even "1" out of q in the internal sequence. In this model, Hell-Johansson and Zhang-Feng attacks correspond to $q = 1$. Our aim in this Section is to find the best tradeoff for $q > 1$.

In order to achieve this, we choose to limit the length of the blocks to a value $k' = 2k$, where $k \geq q$. The probability for a block to have length $2k$ is $\frac{\binom{k-1}{q-1}}{2^k}$, where $\binom{k-1}{q-1}$ is the number of possibilities for the even bits, assuming the first even bit is "1" and there are $q-1$ other "1"s among the even bits of the block. Thus the probability for a block to have length at most $2k$ is:

$$p_{q,k} = \sum_{j=q}^{k} \frac{\binom{j-1}{q-1}}{2^j}$$

If the number of blocks for which we guess the position is l, then the probability for all the blocks to have length at most $2k$ is $(p_{q,k})^l$.

To compute this value l, we need to know the amount of information we have obtained when all the lengths of the blocks are fixed. The entropy leakage provided by the keystream gives q bits of information per block. Then if we call h the amount of information we guess for one block, the total amount of information we then know is $h + q$.

Let us compute $h_{q,k}$, that is h for a block of length $2k$. This information only concerns the even bits of the block. The number of possibilities for the even bits is $\binom{k-1}{q-1}$, *i.e.* the number of manners to distribute the $q-1$ even 1s among the $k-1$ even bits of the subblock made of all the pairs of the block but the first one. This leads to an entropy of $\log(\binom{k-1}{q-1})$. This quantity is the information we still need to guess to have the full knowledge about the even bits of the block, that is k bits of information. Thus the amount we already know (*i.e.* what we have guessed on the even bits) is

$$h_{q,k} = k - \log(\binom{k-1}{q-1})$$

Then for each q we need to find $h_{q,min}$, *i.e.* the minimum of $h_{q,k}$ over all k. We found that for $q = 2$, the minimum of the function holds when $k = 2$, for $q = 3$ when $k = 4$ and for $q = 4$ when $k = 6$. We give the values of these minima in table 4.

Table 4. Minimal information known for the even bits of one block

	$q = 2$	$q = 3$	$q = 4$	$q = 5$
$h_{q,min}$	2	2.415	2.678	2.87

The minimal information we know about one block is $h_{q,min} + q$. We need an information of n bits to recover the key. We still suppose that there is no redundancy in this information. We now can compute the number of blocks l for which we guess the positions, as we know that

$$l(h_{q,min} + q) = n$$

and we obtain $l = \frac{n}{h_{q,min}+q}$.

Our attack is described in algorithm 6.1.

Algorithm 6.1 Our Attack

INPUT : q,k, and a sequence of keystream of length N
OUTPUT: values of the n key bits
PROCESSING:
compute l depending on q and k
For all the k^l possibilities for the length of the l blocks:
    For $j = 0$ to $N - kl$:
        ⋄ Write the system of equations of degree q corresponding to the keystream indexed from x_j
        ⋄ Solve the system of equations by running MiniSAT on it.
        ⋄ Run the SSG forward on the candidate(s) key(s).
        ⋄ If the candidate key is the right one, output it and break the loop.

Now let us compute the amount of keystream necessary for this attack. We have computed the probability that all the l blocks have length at most $2k$, that is $(p_{q,k})^l$. Thus the keystream length N should satisfy $(N - kl) \cdot (p_{q,k})^l \geq 1$ if we want to find at least one match pair between the real internal sequence and our guess. Then we must have:

$$N \geq \frac{1}{(p_{q,k})^l}$$

At each step we try $(k-q+1)^l$ possibilities for the length of the blocks. As the worst case for this attack is a number of steps N, the worst case complexity is:

$$\left(\frac{k-q+1}{\sum_{j=q}^{k}\frac{\binom{j-1}{q-1}}{2^j}}\right)^{\frac{n}{q+h}}$$

where h stands for $h_{q,min}$.

This complexity is true if the information obtained is not redundant. We made simulations by choosing a number of blocks of exactly $\lceil\frac{1}{(p_{q,k})^l}\rceil$ and we always obtained the right key. If the key space given by the SAT solver is larger, we just perform an exhaustive search at small scale.

Now we give the results of our computations. The details of the computations are in appendix C. In this Section, instead of choosing random keys for our simulations, we chose keys such that the blocks in the initial state of the LFSR have length at most k. To achieve this, when generating randomly each block inside the initial state, we test (once it has reached length k) whether the number of "1"s among the even bits is at least q. If not, we start again from the beginning of the block. When the number of "1"s is as expected, we do it for the following block, until we find a compliant key.

We try many such compliant keys in order to limit also the length of the other blocks in the sequence but when k is very small ($k = q + 1$ or $k = q + 2$) we could not achieve the real conditions of the attack due to our limited computational power. Of course the running time would be shorter in the exact case described in the attack as, as we can see it in table 10 and 11 (appendix C), the shorter the blocks are, the faster MiniSAT is for these system of equations.

In table 5 and table 6, we give the total complexities of our attacks, for different block lengths. The Hamming weight of the feedback polynomials are 5 for both LFSR state length 256 and 512. The memory requirements during the MiniSAT computations are never more than 100Mb for this systems.

Finally Table 7 provides a performance comparison between Mihaljević attack, Hell-Johansson attack and our new method, for various sizes of n and of the amount of available keystream. For our attack, the results are bounded by our computational power and would have probably been better if we could have performed all the computations for $q = 4$ and $n = 512$. Anyway the obtained (heuristical) complexities show that for this feedback polynomial Hamming weight, our attack gives the best time/data tradeoff against the self-shrinking generator.

Table 5. Total complexity and data complexity for $n = 256$

	$k=q+1$		$k=q+2$		$k=q+3$		$k=q+4$	
	time	data	time	data	time	data	time	data
$q=2$	$2^{146.2}$	2^{64}	$2^{154.2}$	$2^{34.6}$	$2^{170.9}$	$2^{19.2}$	$2^{181.4}$	$2^{10.7}$
$q=3$	$2^{151.4}$	$2^{79.3}$	$2^{147.2}$	$2^{47.3}$	2^{150}	$2^{28.7}$	$2^{157.2}$	$2^{17.5}$
$q=4$	$2^{153.6}$	$2^{92.6}$	$2^{146.3}$	2^{59}	$2^{147.2}$	$2^{38.3}$	$2^{151.5}$	2^{25}

Table 6. Total time complexity and data complexity for $n = 512$

	$k=q+1$		$k=q+2$		$k=q+3$		$k=q+4$	
	time	data	time	data	time	data	time	data
$q=2$	$2^{279.2}$	2^{128}	$2^{295.7}$	$2^{69.2}$	$2^{318.8}$	$2^{38.3}$	$2^{343.8}$	$2^{21.4}$
$q=3$	$2^{277.4}$	$2^{158.7}$	$2^{269.6}$	$2^{94.6}$	$2^{279.3}$	$2^{57.5}$	$2^{293.5}$	2^{35}
$q=4$	$2^{284.9}$	2^{185}	$2^{278.1}$	$2^{118.1}$	$2^{268.8}$	$2^{76.7}$	$> 2^{293}$	$2^{49.9}$

Table 7. Time complexity comparisons between Mihaljević, Hell *et al.* and our attack for the same data complexities

	$n=256$				$n=512$			
data	$2^{65.3}$	$2^{49.2}$	$2^{39.1}$	$2^{17.5}$	2^{128}	$2^{94.6}$	$2^{57.5}$	$2^{38.6}$
Mihaljević attack	2^{153}	2^{160}	$2^{165.5}$	2^{182}	2^{297}	2^{311}	2^{331}	2^{335}
Hell *et al* attack	$2^{160.2}$	$2^{164.8}$	$2^{167.8}$	$2^{176.4}$	2^{300}	$2^{308.3}$	2^{320}	2^{328}
Our attack	$2^{146.2}$	$2^{147.2}$	$2^{147.2}$	$2^{157.2}$	$2^{268.8}$	$2^{268.8}$	$2^{279.3}$	$2^{293.5}$

7 Conclusion

In [8] and [14], where the best known time/data tradeoffs are proposed on the self-shrinking generator, the authors show that their attack is independent from the value of the Hamming weight of the feedback polynomial defining the LFSR. However, the new algebraic guess-and-determine attack described here suggests that the security of SSG does depend on this Hamming weight. This new attack is very flexible concerning keystream requirement. As we use SAT solvers to solve our algebraic systems, it is not possible to compute a precise time complexity for our attack. However for small Hamming weight values (*i.e.* at most 5), this attack has a noticeably better complexity than the attacks of [8] and [14] and is even the best heuristical time/data tradeoff known so far on the self-shrinking generator.

Since Meier and Staffelbach original paper, avoiding low Hamming weight feedback polynomials has been a widely believed principle. However this rule did not materialize in previous recent attacks. With the new attacks described in this paper, we show explicitly that this principle remains true.

References

1. G. Bard, *Algorithms for Solving Linear and Polynomial Systems of Equations over Finite Fields, with Applications to Cryptanalysis*. Ph.D. Dissertation, University of Maryland, 2007.
2. Gregory V. Bard, Nicolas T. Courtois, and Chris Jefferson: *Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers,* `http://eprint.iacr.org/2007/024`
3. Gregory V. Bard and Nicolas T. Courtois: *Algebraic and Slide Attacks on KeeLoq,* preproceedings of FSE 2008, pp. 89-104.
4. Don Coppersmith, Hugo Krawczyk and Yishay Mansour: *The Shrinking Generator,* In Crypto'93, Springer LNCS 773, pp. 22-39, 1994.
5. Nicolas Courtois, Adi Shamir, Jacques Patarin, Alexander Klimov: *Efficient Algorithms for solving Overdefined Systems of Multivariate Polynomial Equations,* , In Advances in Cryptology, Eurocrypt'2000, Springer LNCS 1807, pp. 392-407, 2000.
6. Jean-Charles Faugère: *A new efficient algorithm for computing Gröbner bases (F_4),* Journal of Pure and Applied Algebra 139 (1999) pp. 61-88. See `www.elsevier.com/locate/jpaa`
7. Jean-Charles Faugère: *A new efficient algorithm for computing Gröbner bases without reduction to zero (F5),* Workshop on Applications of Commutative Algebra, Catania, Italy, 3-6 April 2002, ACM Press.
8. Martin Hell, Thomas Johansson: *Two New Attacks on the Self-Shrinking Generator* IEEE Transactions on Information Theory, Vol. 52, no. 8, pp. 3837- 3843, Aug. 2006.
9. Matthias Krause: *BBD-based Cryptanalysis of Keystream Generators,* In Eurocrypt 2002, Springer LNCS 2332, pp. 222-237, 2002.
10. Hugo Krawczyk: *Practical Aspects of the Shrinking Generator,* in FSE'94, Springer LNCS 809, pp. 45-46, 1993.
11. Cameron McDonald, Chris Charnes and Josef Pieprzyk: *Attacking Bivium with MiniSAT,* `http://eprint.iacr.org/2007/040`
12. Willi Meier and Othmar Staffelbach: *The Self-Shrinking Generator*, In Eurocrypt 94, Springer LNCS 950, pp. 205-214, 1994.
13. Miodrag J. Mihaljević: *A faster cryptanalysis of the self-shrinking generator,* ACISP 1996, Springer LNCS 1172, pp. 182-189, 1996.
14. Bin Zhang, Dengguo Feng: *New Guess-and-determine Attack on the Self-Shrinking Generator,* Asiacrypt 2006, Springer LNCS 4284, pp. 54-68, 2006.

15. Erik Zenner, Matthias Krause, Stefan Lucks: *Improved Cryptanalysis of the Self-Shrinking Generator*, ACISP 2001, Springer LNCS 2119, pp. 21-35, 2001.

A Computation of the Information Rate for the Self-Shrinking Generator

We have seen in Section 2.2 that the information rate that the keystream y reveals on the first m bits of internal sequence z is defined as

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

We have:

$$\begin{aligned} H(Z^{(m)}|Y) &= \sum_{y,z} \text{Proba}(Z^{(m)} = z, Y = y) \log(\text{Proba}(Z^{(m)} = z|Y = y)) \\ &= \sum_{y} \text{Proba}(Y = y) \sum_{z} \text{Proba}(Z^{(m)} = z|Y = y) \times \\ &\qquad \log(\text{Proba}(Z^{(m)} = z|Y = y)) \end{aligned}$$

The self-shrinking generator has the property that for $m \geq 1$ the probability that $C(z)$ is prefix for y for a randomly chosen and uniformly distributed $z \in \{0,1\}^m$ is the same for all keystream y. This implies that

$$\sum_{z} \text{Proba}(Z^{(m)} = z|Y = y) \log(\text{Proba}(Z^{(m)} = z|Y = y))$$

is the same for all y and

$$H(Z^{(m)}|Y) = \sum_{z} \text{Proba}(Z^{(m)} = z|Y = y) \log(\text{Proba}(Z^{(m)} = z|Y = y))$$

Let us call Z^0 the random variable of the first pair of bits of the internal sequence, Z^1 the second pair, etc. We have:

$$H(Z^0|Y) = \sum_{z_0} \text{Proba}(Z^0 = z_0|Y = y) \log(\text{Proba}(Z^0 = z_0|Y = y)) = \frac{3}{2}$$

Let us now show by recursion that $H(Z^{(2k)}|Y) = (\frac{3}{2})^k$.

$$\begin{aligned} H(Z^{(2k)}|Y) = \sum_{z_k,\cdots,z_0} &\text{Proba}(Z^k = z_k, \cdots, Z^0 = z_0|Y = y) \times \\ &\log(\text{Proba}(Z^k = z_k, \cdots, Z^0 = z_0|Y = y)) \end{aligned}$$

And as

$$\begin{aligned}&\text{Proba}(Z^k = z_k, \cdots, Z^0 = z_0 | Y = y) =\\ &\text{Proba}(Z^k = z_k | Z^{k-1} = z_{k-1} \cdots, Z^0 = z_0, Y = y) \times\\ &\qquad\qquad \text{Proba}(Z^{k-1} = z_{k-1} \cdots, Z^0 = z_0 | Y = y)\end{aligned}$$

we have:

$$\begin{aligned}H(Z^{(2k)}|Y) = &\sum_{z_{k-1},\cdots,z_0} \text{Proba}(Z^{k-1} = z_{k-1} \cdots, Z^0 = z_0 | Y = y) \times\\ &\sum_{z_k} \text{Proba}(Z^k = z_k | Z^{k-1} = z_{k-1} \cdots, Z^0 = z_0, Y = y) \times\\ &\log(\text{Proba}(Z^k = z_k | Z^{k-1} = z_{k-1} \cdots, Z^0 = z_0, Y = y))\\ + &\sum_{z_{k-1},\cdots,z_0} \text{Proba}(Z^{k-1} = z_{k-1}, \cdots, Z^0 = z_0 | Y = y) \times\\ &\log(\text{Proba}(Z^{k-1} = z_{k-1}, \cdots, Z^0 = z_0 | Y = y)) \times\\ &\sum_{z_k} \text{Proba}(Z^k = z_k | Z^{k-1} = z_{k-1} \cdots, Z^0 = z_0, Y = y).\end{aligned}$$

We know that

$$\sum_{z_k} \text{Proba}(Z^k = z_k | Z^{k-1} = z_{k-1} \cdots, Z^0 = z_0, Y = y) = 1$$

and by recursion

$$\begin{aligned}&\sum_{z_{k-1},\cdots,z_0} \text{Proba}(Z^{k-1} = z_{k-1} \cdots, Z^0 = z_0 | Y = y) \times\\ &\qquad \log(\text{Proba}(Z^{k-1} = z_{k-1} \cdots, Z^0 = z_0 | Y = y)) = \frac{3}{2}(k-1).\end{aligned}$$

Once the first $k-1$ internal sequence pairs are fixed, let r be the number of 1s among the first bits of the $k-1$ pairs. Let us call y' the keystream sequence where the first r bits of y have been removed. Then the pair Z^k can be seen as the first pair of the internal sequence where $C(Z^k)$ is prefix for y'. Thus:

$$\text{Proba}(Z^k = z_k | Z^{k-1} = z_{k-1} \cdots, Z^0 = z_0, Y = y) = \text{Proba}(Z^k = z_k | Y = y')$$

and the first part of $H(Z^{(2k)}|Y)$ is

$$\sum_{z_k} \text{Proba}(Z^k = z_k | Y = y') \log(\text{Proba}(Z^k = z_k | Y = y')) = \frac{3}{2}.$$

We have obtained $H(Z^{(2k)}|Y) = \frac{3}{2}k$ and $\alpha(2k) = 1 - \frac{3}{2} \cdot \frac{1}{2k} \cdot k = \frac{1}{4}$.

B Equations for the general case

This information can still be divided into three parts:

1. The first two bits of the block are known, this can be described by two linear equations.
2. The fact that the even bits of the subblock made of all the pairs of the block but the first one are all "0" but $q-1$ of them is described by :
   - $\binom{k-1}{q}$ degree q polynomials of the form $x_{2i_0}x_{2i_1}\cdots x_{2i_{q-1}} = 0$ where $2k$ is the length of the block and the x_{2i_j} are even bits of the subblock. This describes the fact that there is at most one "1" among the even bits of the subblock.
   - One equation of degree $q-1$: $\sum x_{i_0}x_{i_1}\cdots x_{i_{q-2}} = 1$, where the $x_{i_0}x_{i_1}\cdots x_{i_{q-2}}$ are all the monomials of degree $q-1$, describing the fact that there are at least $q-1$ "1"s among the even bits of the subblock.
3. The fact that the first keystream bit corresponding to this subblock follows the first even "1" of the subblock is described by $\binom{k-1}{q-1}$ degree q equations of the form $x_{2i_0}x_{2i_1}\cdots x_{2i_{q-2}}(x_{2i_0+1} + e_0) = 0$, the fact that the second keystream bit corresponding to this subblock follows the second even one of the subblock is described by $\binom{k-1}{q-1}$ degree q equations of the form $x_{2i_0}x_{2i_1}\cdots x_{2i_{q-2}}(x_{2i_1+1} + e_1) = 0$, etc, where the $e_0, e_1\cdots, e_{q-2}$ are the keystream bits corresponding to the subblock.

The known information on one block of length $2k$ is completely defined by the equations given above.

C Simulations Details

In tables 8 and 9, we give the complexity of the guess and the data complexity for our attack when the size of the LFSR is 256 or 512.

In tables 10 and 11, we give the time complexity of the MiniSAT solving part of the attack. We first give the running time in seconds, and then we give an estimation of the complexity of the form 2^a for each case to be able to compare our attack with the Hell and Johansson attack of [8]. This means that 2^aE is the running time of the solving, where En^3 would be the running time of the Gaussian elimination in the Hell and Johansson attack on the same machine. Concerning the Mihaljević attack, we just consider that testing the found key (by running the generator on it), is about n operations, where n is the size of the key.

We measured $E \approx 2^{-40}$ hours. With this convention, a running time of one hour corresponds to a complexity of 2^{40}.

Table 8. Complexity of the guess and data complexity for $n = 256$

	$k=q+1$		$k=q+2$		$k=q+3$		$k=q+4$	
	time	data	time	data	time	data	time	data
$q=2$	2^{128}	2^{64}	2^{136}	$2^{34.6}$	$2^{147.2}$	$2^{19.2}$	$2^{159.3}$	$2^{10.7}$
$q=3$	$2^{126.6}$	$2^{79.3}$	$2^{122.2}$	$2^{47.3}$	$2^{123.3}$	$2^{28.7}$	$2^{127.3}$	$2^{17.5}$
$q=4$	2^{128}	$2^{92.6}$	$2^{119.8}$	2^{59}	2^{115}	$2^{38.3}$	2^{114}	2^{25}

Table 9. Complexity of the guess and data complexity for $n = 512$

	$k=q+1$		$k=q+2$		$k=q+3$		$k=q+4$	
	time	data	time	data	time	data	time	data
$q=2$	2^{256}	2^{128}	2^{272}	$2^{69.2}$	$2^{294.3}$	$2^{38.3}$	$2^{318.6}$	$2^{21.4}$
$q=3$	$2^{253.2}$	$2^{158.7}$	$2^{244.4}$	$2^{94.6}$	$2^{246.6}$	$2^{57.5}$	$2^{254.6}$	2^{35}
$q=4$	2^{256}	2^{185}	$2^{239.6}$	$2^{118.1}$	2^{230}	$2^{76.7}$	2^{228}	$2^{49.9}$

Table 10. MiniSAT Computations for $n = 256$

	$k=q+1$		$k=q+2$		$k=q+3$		$k=q+4$	
$q=2$	$< 0.001s$	$2^{18.2}$	$< 0.001s$	$2^{18.2}$	$0.046s$	$2^{23.7}$	$0.015s$	$2^{22.1}$
$q=3$	$0.093s$	$2^{24.8}$	$0.109s$	2^{25}	$0.359s$	$2^{26.7}$	$3.39s$	$2^{29.9}$
$q=4$	$0.171s$	$2^{25.6}$	0.311	$2^{26.5}$	15.6	$2^{32.2}$	$616s$	$2^{37.5}$

Table 11. MiniSAT Computations for $n = 512$

	$k=q+1$		$k=q+2$		$k=q+3$		$k=q+4$	
$q=2$	$0.031s$	$2^{22.2}$	$0.046s$	$2^{23.7}$	$0.078s$	$2^{24.5}$	$0.125s$	$2^{25.2}$
$q=3$	$0.06s$	$2^{29.2}$	$0.17s$	$2^{30.6}$	$22.3s$	$2^{37.7}$	$1641s$	$2^{38.9}$
$q=4$	$1.171s$	$2^{28.9}$	$1308.5s$	$2^{38.5}$	1613	$2^{38.8}$	$> 24h$	$> 2^{45}$