Improved Cryptanalysis of the Self-Shrinking Generator

Erik Zenner [⋆], Matthias Krause, Stefan Lucks [⋆⋆]

Theoretische Informatik
University of Mannheim (Germany)
e-mail: {zenner,krause,lucks}@th.informatik.uni-mannheim.de

Abstract. We propose a new attack on the self-shrinking generator [8]. The attack is based on a backtracking algorithm and will reconstruct the key from a short sequence of known keystream bits. We give both mathematical and empirical evidence for the effectiveness of this attack. The algorithm takes at most $O(2^{0.694L})$ steps, where L is the key length. Thus, our attack is more efficient than previously known key reconstruction algorithms against the self-shrinking generator that operate on short keystream sequences.

1 Introduction

The self-shrinking generator [8] is a keystream generator for the use as a stream cipher. It is based on the shrinking principle [2] and has remarkably low hardware requirements. So far, it has shown considerable resistance against cryptanalysis.

In cryptanalysis of a keystream generator, the attacker is assumed to know a segment of the keystream. The system is considered broken if the attacker can predict the subsequent bits of the keystream with success probability higher than pure guessing. One way to achieve this goal is to reconstruct the initial state of the generator, which allows prediction of the remaining keystream sequence with probability 1.

In this paper, we propose a new attack against the self-shrinking generator. It reconstructs the initial state of the generator from a short keystream sequence, requiring $O(2^{0.694L})$ computational steps. The fastest attack previously known that operates on a short keystream sequence [8] requires $O(2^{0.75L})$ steps. The only attack that has the potential to achieve a better running time [9] needs a much longer keystream sequence.

The paper is organised as follows: In section 2, we give an introduction to both the shrinking and the self-shrinking generator, the former

[⋆] Supported by the LGF Baden-Württemberg
[⋆⋆] Supported by DFG grant Kr 1521/3-1

providing the working principle for the latter. Section 3 surveys some of the previous work on cryptanalysis of the self-shrinking generator.

Sections 4-6 describe our attack and its properties. After giving a description of the algorithm in section 4, we prove the running time to be upper bounded by $O(2^{0.694L})$ steps in section 5. Section 6 provides some supplementary experimental results.

We conclude in section 7 by giving some design recommendations that help in strengthening a self-shrinking generator against our attack.

2 Description of the Cipher

2.1 The Shrinking Generator

In [2, 6], Coppersmith, Krawczyk and Mansour introduced a new pseudo-random keystream generator called the **shrinking generator**. It consists of two linear feedback shift registers (LFSR) A and S, [1] generating the m-sequences $(a_i)_{i\geq 0}$ (denoted as A-sequence) and $(s_i)_{i\geq 0}$ (denoted as S-sequence), respectively. The keystream sequence $(z_j)_{j\geq 0}$ is constructed from these two sequences according to the following selection rule: For every clock i, consider the selection bit s_i. If $s_i = 1$, output a_i. Otherwise, discard both s_i and a_i.

This way, a nonlinear keystream is generated. Even a cryptanalyst who knows part of the keystream sequence can not tell easily which z_j corresponds to which a_i, since the length of the gaps (i.e., the number of a_i that have been discarded) is unknown.

In [2], the shrinking generator is shown to have good algebraic and statistical properties. For a generalisation of some of these results, refer to [10]. Also in [2], a number of algebraic attacks that reconstruct the initial state of A and S are given. Note that all of them require exponential running time in the length $|S|$ of LFSR S.

A probabilistic correlation attack against the shrinking generator is discussed in [4, 11]. The authors give both mathematical and empirical treatment of the necessary computation. The resulting attack reconstructs the initial state of A, requiring an exponential running time in the length $|A|$ of LFSR A. Note that in order to reconstruct the initial state of S, another search is required.

[1] The shrinking principle can be applied to any two binary symmetric sources; it is not restricted to LFSR. All of the algebraic results on the shrinking generator, however, are based on the assumption that LFSR are used as building blocks.

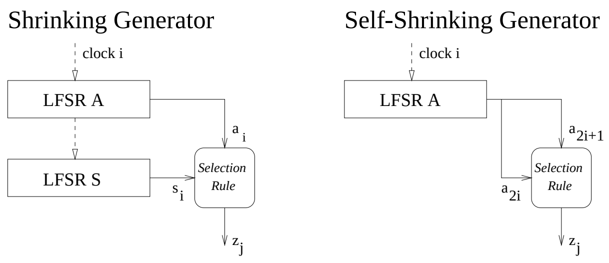


Fig. 1. The Shrinking Generators

As a consequence, a shrinking generator with $|A| \approx |S|$ still remains to be broken by an algorithm that is significantly more effective than the one presented in [2] (for a description, see section 4.1).

2.2 The Self-Shrinking Generator

The **self-shrinking generator** is a modified version of the shrinking generator and was first presented by Meier and Staffelbach in [8].

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

The close relationship between shrinking and self-shrinking generator is shown in figure 1. In [8], an algorithm is given that transforms an L-bit self-shrinking generator into a $2L$-bit shrinking generator. It is also shown that a shrinking generator with register lengths $|A|$ and $|S|$ has an equivalent self-shrinking generator of length $L = 2 \cdot (|A| + |S|)$. Notwithstanding this similarity, the self-shrinking generator has shown even more resistance to cryptanalysis than the shrinking generator. The next section gives a short description of the most efficient key reconstruction attacks that have been proposed in recent years.

3 Previous Work on Cryptanalysis

First, note that the attacks that have been proposed against the shrinking generator can not be transferred to its self-shrinking counterpart. The shrinking generator is best broken by attacking either LFSR A or S, thus effectively halving the key length. The self-shrinking generator, however, has molded both registers into an inseparable unit, namely a single LFSR. For this reason, "separation attacks" can not be employed without major modifications.

3.1 Period and Linear Complexity

The period Π of a keystream sequence generated by a self-shrinking generator was proven to be $2^{\lfloor L/2 \rfloor} \leq \Pi \leq 2^{L-1}$ in [8]. Experimental data seems to indicate that the period always takes the maximum possible value for $L > 3$.

It was also shown that the linear complexity C is always greater than $\Pi/2$. On the other hand, C was proven in [1] to be at most $2^{L-1} - (L-2)$. If $\Pi = 2^{L-1}$, we have $C \in \Theta(2^{L-1})$.

As a consequence, a LFSR with length equal to C can be constructed from about 2^L keystream bits in $O(2^{2L-2})$ computational steps, using the Berlekamp-Massey algorithm [7]. For realistic generator sizes of $L > 100$, this attack is thus computationally unfeasible.

3.2 Attacks using short keystream sequences

Even if the feedback logic of the LFSR is not known, there is a simple way of reducing the key space [8]. Consider the first two bits (a_0, a_1) of the LFSR (unknown) and the first bit z_0 of the keystream (known). Then there are only three out of four possible combinations (a_0, a_1) that are consistent with the keystream, since $(a_0, a_1) = (1, \bar{z}_0)$ is an immediate contradiction. The same rule can be applied for the next bit pair (a_2, a_3), and so on. Consequently, only

$$3^{L/2} = 2^{(log_2(3)/2) \cdot L} = 2^{0.79L}$$

possible initial values for the LFSR A consistent with the keystream.

The running time that is needed to search through the reduced key space can be further reduced on average by considering the likelihood of the keys. Note that the following holds:

$$\begin{aligned} \Pr[(a_0, a_1) = (0,0) | z_0] &= 1/4 \\ \Pr[(a_0, a_1) = (0,1) | z_0] &= 1/4 \\ \Pr[(a_0, a_1) = (1, z_0) | z_0] &= 1/2. \end{aligned}$$

Thus, the entropy of the bit pair is

$$H = -(1/4)\log(1/4) - (1/4)\log(1/4) - (1/2)\log(1/2) = 3/2.$$

The total entropy of an initial state consisting of L/2 such pairs is thus $2^{0.75L}$. At the same time, this is the effort for searching the key space if the cryptanalyst starts with the most probable keys. Surprisingly, this is still the most efficient reconstruction algorithm using short keystream sequences that has been published.

3.3 Attack using long keystream sequences

In [9], Mihaljević presented a faster attack that needs, however, a longer part of keystream sequence. Let the length of this known part be denoted by N. Then the attacker assumes that an l-bit section of the keystream has been generated by the current inner state of the LFSR. Consequently, l out of the $L/2$ even bits of A must be equal to 1. The attacker guesses these bits and checks whether or not this guess can be correct, iterating over all l-bit sections of the keystream. It is shown that cryptanalysis is successful with high probability after 2^{L-l} steps.

Since this procedure only makes sense for $L/4 \leq l \leq L/2$, the running time can vary from $2^{0.5L}$ in the very best case to $2^{0.75L}$ under more unfavourable circumstances. The efficiency of the attack depends mainly on the number of keystream bits that are available, since the value l must be chosen such that the following inequality holds:

$$N > l \cdot 2^{L/2} \cdot \binom{L/2}{l}^{-1}$$

In order to get a feeling for the number of bits required for this attack, table 1 gives some examples of required bitstream lengths for different register sizes L. The number of bits is given in logarithmic form in order to enhance readability. We concentrate on three cases:

- In order to beat the best key reconstruction algorithm described above, we need $l = 0.25L$, yielding a running time of $2^{0.75L}$ steps.
- Improving the running time to $2^{0.694L}$ (which is the performance of the algorithm to be presented in section 4) requires $l = 0.306L$.
- In order to achieve the best possible running time of $2^{0.5L}$ steps, we need $l = 0.5L$. Note that for realistic register lengths, the sheer amount of required data (namely, $N > \frac{L}{2} \cdot 2^{L/2}$) should make such an attack a mere theoretical possibility.

value l:	$0.25L$	0.306	$0.50L$
Time:	$2^{0.75L}$	$2^{0.694L}$	$2^{0.5L}$
Bits :			
$L = 120$	$2^{8.19}$	$2^{10.17}$	$2^{65.91}$
$L = 160$	$2^{8.81}$	$2^{11.37}$	$2^{86.32}$
$L = 200$	$2^{9.30}$	$2^{13.07}$	$2^{106.64}$
$L = 240$	$2^{9.69}$	$2^{14.03}$	$2^{126.91}$
$L = 280$	$2^{10.02}$	$2^{14.94}$	$2^{147.13}$
$L = 320$	$2^{10.31}$	$2^{15.81}$	$2^{167.32}$

Table 1. Number N of keystream bits required for Mihaljević attack

4 The Backtracking Algorithm

The goal of our cryptanalysis is the reconstruction of an inner state of the generator that is consistent with the keystream. We assume thus that a short keystream sequence of length $\approx L$ bits is known to the attacker.

We also assume that the feedback polynomial of the generator is known. Note that none of the attacks given in section 3.2 makes use of the feedback logic. It can be expected that the use of additional information should lead to a more efficient attack.

4.1 Basic Idea: Attacking the Shrinking Generator

First, consider cryptanalysis of the shrinking generator. If the feedback polynomials are known, an obvious way of reconstructing the inner states is as follows.

1. Guess the inner state of the control register S. From this, we can determine as many bits of the S-sequence as required.
2. Knowing the S-Sequence and part of the keystream sequence, we can reconstruct single bits of the A-Sequence.
3. Each known bit of the A-sequence gives a linear equation. If we can find $|A|$ linear independent equations, we can solve the system and thus reconstruct the inner state of register A.
4. We run the shrinking generator, using the reconstructed inner states for A and S. If the keystream sequence thus generated matches the known keystream sequence in the first $|A| + |S| + \epsilon$ positions (where ϵ is a security margin), we have found with high probability the correct inner state.

The running time of this attack (that was also presented in [2]) is obviously upper bounded by $O(|A|^3 \cdot 2^{|S|})$, since there are at most $2^{|S|} - 1$

inner states of register S and the solving of a system of $|A|$ linear equations takes at most $|A|^3$ steps.

4.2 Applying the Idea to the Self-Shrinking Generator

The principle of guessing only the S-Bits and deriving the A-Bits by solving a system of linear equations can be applied to the self-shrinking generator as well. It is, however, not as straightforward as with the shrinking generator, since guessing all S-Bits in the initial state (i.e., all even bits) will not enable the cryptanalyst to compute the rest of the S-sequence (unless the generator has a non-primitive characteristic polynomial). Thus, we will guess the even bits one at a time, using a backtracking approach similar to the procedure proposed by Golić in [3] for cryptanalysis of the A5/1 stream cipher.

Before we describe the details of the attack, we give the following property of the key (i.e. the initial state of the LFSR)[2]:

Proposition 1. *For each key $K = (a_0, \dots, a_{L-1})$ with $a_0 = 0$, there exists an equivalent key $K' = (a'_0, \dots, a'_{L-1})$ with $a'_0 = 1$.*

Proof. Consider the sequence $(a_i)_{i\geq 0}$ generated by the inner state K. Suppose the first '1' on an even position appears in position $2k$. Then clock the register by $2k$ steps, deriving the new inner state $K' = (a_{2k}, \dots, a_{2k+L-1})$. Obviously, both inner states yield the same keystream sequence, since in transforming K to K', no output is generated. $\square$

It is thus safe to assume that $a_0 = 1$ and $a_1 = z_0$. This way, we will reconstruct a key that is not necessarily equal to the original key, but it is equivalent in a sense that it will create the same keystream sequence.

From now on, we will have to guess the even bits of the sequence $(a_i)_{i\geq 0}$. This way, we obtain two different types of equations as follows:

– Every guess can be represented by a linear equation $a_{2i} = b_i$. These equations will be referred to as being of *type 1.*
– If $a_{2i} = 1$, we obtain a second equation of the type $a_{2i+1} = z_j$, where $j = \sum_{c=0}^{i} a_{2c}$. These equations will be denoted as being of *type 2.*

This approach will be implemented using a tree of guesses as shown in figure 2.

[2] The same property also holds for the shrinking generator. In this context, it was discussed in [11].

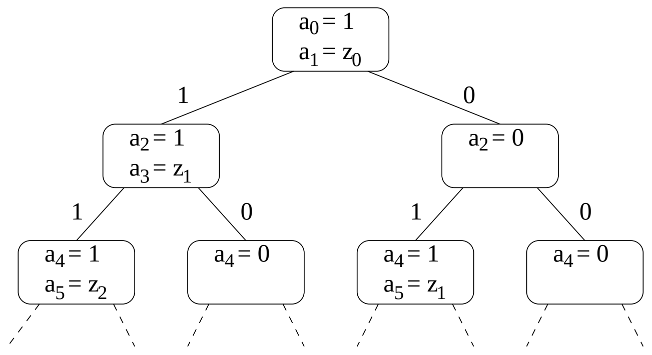


Fig. 2. The Tree of Guesses

As long as $i \leq \lfloor L/2 \rfloor - 1$, the development of the tree is straightforward. We get exactly two new equations whenever we follow a '1' branch and exactly one new equation when following a '0' branch. All of these equations are linearly independent, since no variable a_k appears more than once. Thus, we get a complete binary tree with height $\lfloor L/2 \rfloor - 1$.

After that point, however, the tree becomes irregular, since the indices of the new equations become larger than $L - 1$. Thus, the feedback recurrence must be used to convert the simple equations into a representation using only $a_0, \ldots, a_{L-1}$. Depending on the equations that are already known, there is an increasing probability that the new equations are linearly dependent of the earlier ones. That means they are either useless (in case they are consistent with the existing equation system) or lead to a contradiction. In the latter case, we have chosen a path in the tree that is not consistent with the known keystream sequence. We can thus ignore the current branch and start backtracking.

If we find a branch that ultimately gives us L linearly independent equations, we can solve the equation system and derive a key candidate. This candidate is evaluated by running the self-shrinking generator with this initial value, generating a candidate keystream of length $L + \epsilon$ (where ϵ is a small number of additional bits). We compare the candidate keystream with the known keystream segment. If they match, the key candidate is equivalent to the original key with high probability.

5 Upper Bounding the Running Time

In this section, we establish an asymptotical upper bound on the running time of our algorithm. For this purpose, we first give an upper bound C_L

for the number of *leaves* in the tree of guesses (sections 5.1-5.3). Then, in section 5.4 we derive an upper bound for the number N_L of *nodes* in the tree and conclude that the total running time of the algorithm can be upper bounded by $O(L^4 \cdot 2^{0.694L})$.

5.1 Well-formed vs. malformed trees

Let T_ℓ denote a tree of guesses such that ℓ linearly independent equations are still missing in the root to allow the solving of the equation system. Note that for the search tree given in section 4, we have $\ell = L - 2$.

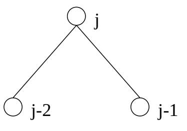
In order to formally prove the maximum number C_ℓ of leaves in T_ℓ, we label the nodes as follows: Each node is labelled by the number of linearly independent equations still needed in order to solve the equation system. The root is thus labelled by ℓ. For technical reasons, we allow a leaf of the tree to take both the labels 0 and -1, both meaning that the system is completely specified.

Assumption 1 *For the following average case analysis, we assume that an equation that is linearly dependent of its predecessors will lead to a contradiction with probability 1/2.*

This assumption is reasonable, since the bits a_{2i} and a_{2i+1} are generated by an m-LFSR, meaning that a variable takes values 0 and 1 with (almost) equal probability.

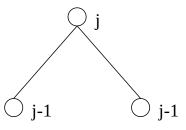
Now consider an arbitrary node V of depth $i-1$, $i \geq 1$, and its two children, V_0 and V_1 (reached by guessing $a_{2i} = 0$ or $a_{2i} = 1$, resp.) Let V be labelled by j. The labelling of the child nodes depends on whether a_{2i} and/or a_{2i+1} are linearly dependent of the previous equations or not:

A) *Both are independent.* In this case, no contradiction occurs. The left child is labelled $j-2$ and the right child is labelled $j-1$.

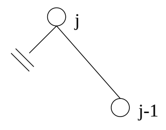


Prob = 1

B) a_{2i} *is independent,* a_{2i+1} *is not.* Both children are labelled $j-1$. However, a contradiction occurs in V_1 with probability of 1/2.

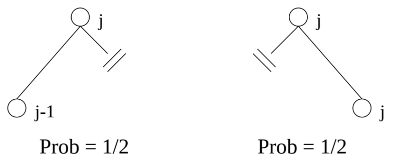


Prob = 1/2

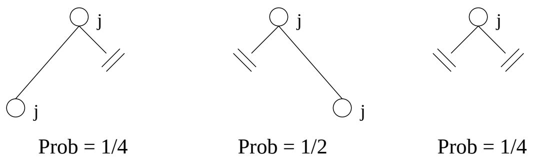


Prob = 1/2

C) *a_{2i} is dependent, a_{2i+1} is not.* The left child is labelled $j-1$, while the right child is labelled j. However, a contradiction occurs either in V_1 or in V_0, with equal probability.



D) *Both are dependent.* In this case, both child nodes have the same label as the parent node. Due to the linear dependency of a_{2i} there occurs a contradiction in either V_1 or V_0, with equal probability. In addition, there is an additional probability of $1/2$ that a_{2i+1} leads to a contradiction in V_1.



Definition 1. *A **well-formed tree** T^*_ℓ is a binary tree where only branchings of type A occur, i.e., for every node that is not a leaf, the following rule holds: If the label of the node is j, then the label of its left child is $j-2$ and the label of its right child is $j-1$.*
*A **malformed tree** is an arbitrary tree of guesses that contains at least one branching of a type B, C or D.*

Essentially, the notion of a well-formed tree describes the tree of guesses under the assumption that all linear equations (of both type 1 and 2) are linearly independent. Note that such a tree is highly unlikely for large ℓ. Nonetheless, the well-formed tree plays an important role in establishing the overall number of leaves for the tree of guesses. We proceed now to prove that on average, a malformed tree has at most the same number of leaves as a well-formed tree.

Theorem 1. *Let C^*_ℓ denote the number of leaves of a well-formed tree T^*_ℓ. Let C_ℓ denote the maximum number of leaves in a tree T_ℓ that may or may not be malformed. Then in the average case, $C_\ell \le C^*_\ell$ holds.*

Proof. The proof is by induction. Obviously, the inequality holds for C_{-1} and C_0, since trees T_{-1} and T_0 consist only of a root without a child.

Thus, $C_{-1} = C^*_{-1} = 1$ and $C_0 = C^*_0 = 1$.
Now consider C_ℓ, $\ell \geq 1$. First note that since the theorem holds for $C_{\ell-1}$ and $C_{\ell-2}$, it follows that

$$C_{\ell-1} + C_{\ell-2} \leq C^*_{\ell-1} + C^*_{\ell-2} = C^*_\ell. \tag{1}$$

Also note that even in the worst possible branching case, we have

$$C_\ell \leq 2 \cdot C_{\ell-1}. \tag{2}$$

for all ℓ. Using these two facts, we can prove an upper bound for C_ℓ by distinguishing the following cases (identical to the ones given above):

A) Let the tree T^A_ℓ be composed of a subtree with at most $C_{\ell-2}$ leaves and a subtree with at most $C_{\ell-1}$ leaves. It follows for the maximum number C^A_ℓ of such a tree that

$$C^A_\ell \leq C_{\ell-2} + C_{\ell-1} \leq C^*_\ell.$$

B) The tree T^B_ℓ is composed of either one or two subtrees, having at most $C_{\ell-1}$ leaves each. Consequently, $C^B_\ell \leq 1/2 \cdot C_{\ell-1} + C_{\ell-1}$. Using (2), we have

$$C^B_\ell \leq C_{\ell-2} + C_{\ell-1} \leq C^*_\ell.$$

C) The tree T^C_ℓ is composed of only one subtree with at most $C_{\ell-1}$ or C_ℓ leaves, resp. (with equal probability). We have $C^C_\ell \leq 1/2 \cdot (C_{\ell-1} + C_\ell)$, and using (2), derive

$$C^C_\ell \leq \frac{1}{2}(2C_{\ell-2} + 2C_{\ell-1}) = C_{\ell-2} + C_{\ell-1} \leq C^*_\ell.$$

D) The tree T^D_ℓ has one of the forms given in case D. Then, for the average number C^D_ℓ of leaves in this tree, we have $C^D_\ell \leq \frac{3}{4} \cdot C_\ell$. Using (2) repeatedly, we get

$$C^D_\ell \leq \frac{3}{2} \cdot C_{\ell-1} = C_{\ell-1} + \frac{1}{2}C_{\ell-1} \leq C_{\ell-1} + C_{\ell-2} \leq C^*_\ell.$$

Since $C_\ell = \max(C^A_\ell, C^B_\ell, C^C_\ell, C^D_\ell)$, we have $C_\ell \leq C^*_\ell$. □

5.2 Size of a well-formed tree

We have shown that the number C_ℓ of leaves in an arbitrary tree of guesses is on average not bigger than the number C^*_ℓ of leaves in a well-formed tree. In the next section, we will prove an estimate for C^*_ℓ and thus an upper bound for C_ℓ.

Theorem 2. *Let C_ℓ^* denote the size of a well-formed tree T_ℓ^*. Then we have $a^\ell \leq C_\ell^* \leq \frac{2}{a}a^\ell$ for all $L \geq 1$, where $a = \frac{1+\sqrt{5}}{2} \approx a^{0.6942419}$.*

Proof. Note that for all $\ell \geq -1$, C_ℓ^* satisfies the recursion $C_{\ell+2}^* = C_{\ell+1}^* + C_\ell^*$ with $C_{-1}^* = C_0^* = 1$.
Let a be the unique positive solution of $x^2 = x + 1$, i.e. $a = \frac{1+\sqrt{5}}{2}$. In this case, the function $F(\ell) = a^\ell$ also satisfies the recursion $F(\ell+2) = F(\ell+1) + F(\ell)$ for all $\ell \geq 0$. Since $C_0 = F(0)$ and $C_1 = \frac{2}{a}F(1)$, we have $a^\ell \leq C_\ell \leq \frac{2}{a}a^\ell$ for all $\ell \geq 0$. □

Note that $\frac{2}{a} \approx 1.236068$. Thus, we have found the upper bound of the average search tree to be $C_\ell \leq \frac{2}{a} \cdot 2^{0.694\ell} \approx 2^{0.694\ell+0.306}$.

5.3 Worst case considerations

The above result can be applied directly to the tree of guesses in section 4. Remembering that such a search tree actually has a root labelled $\ell = L-2$, we can upper bound the average number of leaves by $C_L \leq 2^{0.694L-0.918}$.

This upper bound seems to holds even for the worst case, provided that L is large enough. Remember that assumption 1 stated that in case of a linearly dependent equation, contradiction occurs with probability $1/2$. Now remember from section 4.2 that linearly dependent equations do not occur before depth $\lfloor\frac{L}{2}\rfloor$ is reached. This, in turn, means that for large L, there exists a large number of nodes labelled j for each $j < L - \lfloor\frac{L}{2}\rfloor$. We can thus apply the law of large numbers, stating that the actual number of contradictions is very close to the expected number of contradictions. Thus, the number of leaves should be close to the above bound not only for the average case, but for almost any tree of guesses.

In order to give some more weight to this rather informal argument, we will provide some empirical evidence for this conjecture in section 6.

5.4 Running time of the algorithm

The asymptotically most expensive single step of the backtracking algorithm presented in section 4 is the testing of the linear dependency of new equations. This operation in itself takes $O(L^3)$ elementary steps and has to be repeated once or twice in each node of the tree of guesses.

Thus, we have to establish an upper bound for the maximum number of *nodes* in the tree. Since the tree will be malformed, it contains nodes that have only one child. It is thus impossible to upper bound the number of nodes by $2 \cdot C_L - 1$, as could be done for a proper binary tree. We can, however, prove that the maximum depth of the search tree is $L - 1$.

Proposition 2. *If the linear recurrent sequence $(a_i)_{i\geq 0}$ is of maximum length, then the tree has maximum height of $L-1$.* [3]

Proof. In any node of depth i, we have exactly $i+1$ equations of type 1 at our disposal (and a varying number of equations of type 2). Thus, at depth $L-1$, we have exactly L such equations, namely $a_0, a_2, \ldots, a_{2L-2}$. By a theorem on maximum length linear recurrent sequences (see e.g. [5], p. 76), there exists a k such that the following holds:

$$(a_k, a_{k+1}, \ldots, a_{k+L-1}) = (a_0, a_2, \ldots, a_{2L-2})$$

Since $a_k, \ldots, a_{k+L-1}$ must be linearly independent, the same holds for $a_0, a_2, \ldots, a_{2L-2}$. Consequently, we have L linearly independent equations of type 1 in any node of depth $L-1$, allowing us to solve the system and derive a key candidate. Thus, no node of the tree will have depth $\geq L$. $\square$

We can use this fact to upper bound the number of nodes. Consider the largest binary tree (w.r.t. the number of nodes) with height $L-1$ and C_L leaves. This tree is a complete binary tree from depth 0 to $p = \lfloor \log C_L \rfloor$. From depth $p+1$ to depth $L-1$, the tree has constant width of C_L.

Let N_L denote the number of nodes in a search tree. It follows that N_L is at most the size of this worst possible tree.

$$N_L \leq (2^{p+1} - 1) + (L - p - 1) \cdot C_L$$

Note that both 2^{p+1} and C_L are in $O(C_L)$. Ignoring all constant summands and factors to C_L, we obtain:

$$\begin{aligned} N_L &\in O((L-p) \cdot C_L) \\ &= O(0.306L \cdot 2^{0.694L - 0.918}) \\ &= O(0.162L \cdot 2^{0.694L}) \end{aligned}$$

Remembering that in each node, a linear equation has to be inserted into an equation system, and ignoring constant factors again, we derive a total asymptotic running time in $O(L^4 \cdot 2^{0.694L})$.

[3] Note that this proposition only holds for maximum length sequences. The use of shorter sequences, however, would be a breach of elementary design principles, since it would facilitate a number of other attacks. It does not seem to increase resistance against our attack either, it just makes the proof harder.

	Number of leaves			Number of nodes		
L	C_{avg}	C_{max}	C_{bound}	N_{avg}	N_{max}	N_{bound}
3	$2^{1.00}$	$2^{1.00}$	$2^{1.16}$	$2^{1.58}$	$2^{1.58}$	$2^{1.04}$
4	$2^{1.55}$	$2^{1.58}$	$2^{1.86}$	$2^{2.57}$	$2^{2.81}$	$2^{2.15}$
5	$2^{2.29}$	$2^{2.58}$	$2^{2.55}$	$2^{3.28}$	$2^{3.46}$	$2^{3.17}$
6	$2^{2.85}$	$2^{3.17}$	$2^{3.25}$	$2^{4.21}$	$2^{4.64}$	$2^{4.12}$
7	$2^{3.53}$	$2^{3.81}$	$2^{3.94}$	$2^{4.92}$	$2^{5.43}$	$2^{5.04}$
8	$2^{4.23}$	$2^{4.64}$	$2^{4.63}$	$2^{5.61}$	$2^{5.93}$	$2^{5.93}$
9	$2^{4.88}$	$2^{5.29}$	$2^{5.33}$	$2^{6.35}$	$2^{6.79}$	$2^{6.79}$
10	$2^{5.53}$	$2^{5.88}$	$2^{6.02}$	$2^{7.05}$	$2^{7.55}$	$2^{7.64}$
11	$2^{6.22}$	$2^{6.57}$	$2^{6.72}$	$2^{7.75}$	$2^{8.24}$	$2^{8.47}$
12	$2^{6.87}$	$2^{7.26}$	$2^{7.41}$	$2^{8.46}$	$2^{8.89}$	$2^{9.29}$
13	$2^{7.56}$	$2^{7.92}$	$2^{8.10}$	$2^{9.16}$	$2^{9.73}$	$2^{10.10}$
14	$2^{8.25}$	$2^{8.56}$	$2^{8.80}$	$2^{9.85}$	$2^{10.20}$	$2^{10.90}$
15	$2^{8.92}$	$2^{9.23}$	$2^{9.49}$	$2^{10.56}$	$2^{11.26}$	$2^{11.69}$
16	$2^{9.61}$	$2^{9.90}$	$2^{10.19}$	$2^{11.25}$	$2^{11.64}$	$2^{12.48}$

Table 2. Empirical Results

6 Experimental Results

6.1 Results on the number of leaves

In the section 5, we have proven the number of leaves in the search tree to be upper bounded by $2^{0.694L-0.918}$ in the average case. This result leaves a number of interesting questions open: Since we have only derived an upper bound: How close is this value to the average number of leaves that do occur in an actual search?[4] And what about the conjecture in section 5.3? Is C_L also an upper bound for the worst case, for large L?

In order to answer those questions, the key reconstruction algorithm from section 4 has been implemented and tested against all keys and all primitive polynomials for $L = 3, \dots, 16$. The main results of this simulation are given in the left half of table 2. Here, C_{avg} and C_{max} denote the average and maximum number of leaves encountered in the experiments. $C_{bound} = 2^{0.694L-0.918}$ denotes the upper bound as calculated in section 5. For ease of comparison, all values are given in logarithmical notation.

First observe that values C_{avg} and C_{max} are very close; they differ by a factor ϕ with $1 < \phi < 1.33$. Of course, this may or may not hold for larger values of L, but for small L, the maximum number of leaves does not stray very far from the average.

[4] We must take care not to confuse the average case of the analysis with the average number of leaves in the search tree; they are quite different mathematical objects.

Also observe that for $L > 8$, C_{bound} seems to be a proper upper bound not only for the average case, but also for the maximum number of leaves in the search tree. Note especially that for $L > 8$, the gap between C_{max} and C_{bound} seems to be widening with increasing L. Nonetheless, additional empirical or mathematical evidence for larger L might be necessary before our conjecture from section 5.3 can be considered confirmed.

6.2 Results on the number of nodes

In the right half of the table, we give the results on the number of nodes. Again, N_{avg} and N_{max} denote the average and maximum values encountered in the experiments, while $N_{bound} = 0.162L \cdot 2^{0.694L}$ denotes the mathematical bound as given in section 5.4.

It seems that for $L > 7$, N_{bound} is an upper bound for the number of nodes in the worst possible case. As with the results on the number of leaves, the gap between N_{max} and N_{bound} seems to be widening with increasing L, but again, more data for larger L would be helpful. We also note that N_{avg} and N_{bound} are very close to each other.

An interesting side observation is that $N_{avg} \approx 2 \cdot C_{bound}$, i.e. that the average number of nodes appears to be almost exactly twice the mathematical upper bound for the number of leaves as derived in section 5.3. This is not apparent from the mathematical analysis in section 5 and may thus be an interesting starting point for future research.

7 Design Recommendations

The effective key size against our attack is less than 70% of the key length. For a register length of 120 bit, the backtracking attack runs in $O(2^{83})$ steps and is probably not feasible in today's practice. Our attack, however, is easily parallelised, allowing an adversary to use as many parallel processors at once as he can afford. Since each processor can operate on its own segment of the tree (without any need of communication with the other ones), k processors can reduce the running time by a factor of k. Thus, a generator using a shorter register is in real danger of being compromised. We conclude that the **minimum length** of a self-shrinking generator should exceed 120 bit.

Note that our attack relies on the feedback logic of the register to be known. If it is not, the attack has to be repeated for all primitive feedback polynomials of length L, yielding an additional working factor of $\phi(2^L - 1)/L$. Security of the self-shrinking generator can thus be increased significantly by following the proposal given in [2, 8]: Use a **programmable feedback logic** and make the actual feedback polynomial a part of the key.

Finally, observe that the use of sparse feedback polynomials makes our attack slightly more effective. If the more significant bits depend on only a few of the less significant bits, the probability of linear dependent equations increases, yielding a tree of guesses that is more slender than the average case tree considered above. However, as stated in section 6.2, the sizes of worst case and best case trees seem to differ by less than the factor 2. Nonetheless, **sparse feedback polynomials** should be avoided in designing most stream ciphers, the self-shrinking generator being no exception.

Acknowledgement We would like to thank one of the anonymous referees for a number of very helpful comments.

References

1. S.R. Blackburn. The linear complexity of the self-shrinking generator. *IEEE Transactions on Information Theory*, 45(6):2073–2077, September 1999.
2. D. Coppersmith, H. Krawczyk, and Y. Mansour. The shrinking generator. In D.R. Stinson, editor, *Advances in Cryptology - EUROCRYPT '93*, volume 773 of *LNCS*, pages 22–39, Berlin, 1993. Springer-Verlag.
3. J.D. Golić. Cryptanalysis of alleged A5 stream cipher. In W. Fumy, editor, *Advances in Cryptology - EUROCRYPT '97*, volume 1233 of *LNCS*, pages 239–255, Berlin, 1997. Springer-Verlag.
4. J.D. Golić and L. O'Connor. Embedding and probabilistic attacks on clock-controlled shift registers. In A. De Santis, editor, *Advances in Cryptology - EUROCRYPT '94*, volume 950 of *LNCS*, pages 230–243, Berlin, 1995. Springer-Verlag.
5. S.W. Golomb. *Shift Register Sequences.* Aegean Park Press, Laguna Hills (CA), revised edition, 1982.
6. H. Krawczyk. The shrinking generator: Some practical considerations. In R. Andersen, editor, *Fast Software Encryption '93*, volume 809 of *LNCS*, pages 45–46, Berlin, 1994. Springer-Verlag.
7. J.L. Massey. Shift register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15:122–127, 1969.
8. W. Meier and O. Staffelbach. The self-shrinking generator. In A. De Santis, editor, *Advances in Cryptology - EUROCRYPT '94*, volume 950 of *LNCS*, pages 205–214, Berlin, 1995. Springer-Verlag.
9. M.J. Mihaljević. A faster cryptanalysis of the self-shrinking generator. In J. Pieprzyk and J. Seberry, editors, *Advances in Cryptology - ACISP '96*, volume 1172 of *LNCS*, pages 182–189, Berlin, 1996. Springer-Verlag.
10. I. Shparlinski. On some properties of the shrinking generator. `http://www.comp.mq.edu.au/~igor/Shrink.ps`.
11. L. Simpson, J.D. Golić, and E. Dawson. A probabilistic correlation attack on the shrinking generator. In C. Boyd and E. Dawson, editors, *Advances in Cryptology - ACISP '98*, volume 1438 of *LNCS*, pages 147–158, Berlin, 1998. Springer-Verlag.