Fault Cryptanalysis and the Shrinking Generator^{*}

Marcin Gomułkiewicz¹, Mirosław Kutyłowski¹, and Paweł Wlaz²

¹ Wrocław University of Technology

² Lublin University of Technology

Abstract. We present two efficient and simple fault attacks on the shrinking generator. In a first case if the attacker can stop control generator for some small number of steps and observe the output, then with high probability he can deduce the full control sequence, and so the other input bitstream. The second method assumes that the attacker can disturb the control sequence (in an unpredictable and random way) and observe many samples of such experiments. Then he can reconstruct a certain sequence that agrees with the input sequence of the generator on a large fraction of bits.

1 Introduction

This paper presents two new fault attacks (see [2]) against the shrinking generator by Coppersmith *et al.* [3], one of the major designs of efficient and secure pseudorandom generators. Our attacks are unique in many ways. To our best knowledge these are the first fault attacks on the shrinking generator; they also seem to be among the most efficient attacks. On top of that, it is generally agreed that the shrinking generator is composed out of LFSRs. However, we require no specific design of the generator (it can consist of any bit generators), apart from the possibilities regarding injecting faults of the assumed kind.

The paper is organized as follows: first we give a very brief overview of the shrinking generator. Then the Section 2 describes our first attack, based on synchronization fault assumption (the assumption is similar to those from [9]). The Section 3 presents another attack, based on weaker assumptions. Finally the Section 4 briefly summarizes our results and states open problems.

The Shrinking Generator in a Pill The shrinking generator [3] is an attempt to create cryptographically strong pseudorandom bitstream generator out of relatively weak components. Many other solutions of this kind [7, 1, 4] were proven to be weak [13, 14]. The shrinking generator successfully faces the trial of time: the best known attacks against it are exponential in the LFSR's length [5, 8, 10–12], or based on the assumption that the feedback is known [6].

^{*} Partially supported by KBN project no. 0T00A 003 23. To be published by Springer-Verlag, LNCS 4007, copyright Springer-Verlag

Amazingly, the construction of the shrinking generator is very simple. It consists of two LFSRs we shall call the base (or input) generator A and the control generator C ; their output is denoted as $a_1, a_2, a_3, \dots$ and $c_1, c_2, c_3, \dots$, respectively. The output $z_1, z_2, z_3, \dots$ is composed of those and only those of a_i for which $c_i = 1$. Formally: $z_t = a_i$ for i so that:

$$t = \sum_{j=1}^i c_j \quad \text{and} \quad c_i = 1 .$$

2 An Attack with Synchronization Faults

Assumptions The attacker has a device implementing the shrinking generator and can use it freely. Assume that the base and control generators of the shrinking generator output bits according to the uniform distribution over $\{0, 1\}$. Also assume, that an attacker can disturb clocking of the device, namely we assume that he can stop the control sequence for a couple of steps, and observe the generator's output.

2.1 Basic Attack

Assume we have number of output sequences related to holding the control generator for 0 (i.e. with no fault), $1, 2, \dots, n - 1$ steps. That is, we stop the control generator for a number of steps, the base generator moves on, so we have sequence of outputs $Z^k = z_1^k z_2^k \dots z_N^k$ related somehow to the base bitstreams $a_{k+1}, a_{k+2}, a_{k+3}, \dots$ and the control bitstream $c_1, c_2, c_3, \dots$.

Let us take a closer look at the data – see Table 1. Our algorithm shall guess the length of blocks of zeroes that separate consecutive ones in the control bitstream. Note that if two ones are consecutive, the 2nd bit of the bitstream i is the same as the 1st bit of the bitstream $i + 1$, for each i . Analogously, if the ones are separated by a single zero, then the 2nd bit of bitstream i is the same as the 1st bit of bitstream $i + 2$, for each i . If so, it is easy to construct an algorithm that shall guess the number of zeroes separating the ones. It returns a set S_l of all non-contradictory solutions – number of zeroes in the control stream between the two ones corresponding to output positions l and $l + 1$. However, we have to assume that the number of zeroes between two ones does not exceed a certain parameter *maxzeroes*. To obtain the whole control sequence we have to execute the algorithm for $l = 1, 2, \dots, N - 1$ and consider the set of all possible solutions $\mathbf{S} = S_1 \times S_2 \times \dots \times S_{N-1}$. Then $\mathbf{s} = (s_1, s_2, \dots, s_{N-1})$ corresponds to a possible control sequence:

$$\underbrace{00 \dots 0}_? \underbrace{100 \dots 0}_{s_1} \underbrace{100 \dots 0}_{s_2} 1 \dots 1 \underbrace{00 \dots 0}_{s_{N-1}} 1 .$$

Of course, it is impossible to recover the number of zeroes preceding the first one in the control sequence.

A	a_i	a_{i+1}	a_{i+2}	a_{i+3}	a_{i+4}	a_{i+5}	a_{i+6}
S	1	1	0	1	0	0	1
Z^0	a_i	$\dagger a_{i+1}$	a_{i+2}	$*a_{i+3}$	a_{i+4}	a_{i+5}	$\ddagger a_{i+6}$
Z^1	$\dagger a_{i+1}$	a_{i+2}	a_{i+3}	a_{i+4}	a_{i+5}	a_{i+6}	a_{i+7}
Z^2	a_{i+2}	$*a_{i+3}$	a_{i+4}	a_{i+5}	a_{i+6}	a_{i+7}	a_{i+8}
Z^3	a_{i+3}	a_{i+4}	a_{i+5}	$\ddagger a_{i+6}$	a_{i+7}	a_{i+8}	a_{i+9}
$\dots$				$\dots$			

Table 1. Idea of the basic attack

Probability of a False Solution

Lemma 1. *Let n be the number output sequences analyzed and the correct number of zeroes between consecutive ones considered is l . Then the output set S contains l with probability 1, and contains an m , $m \neq l$, with probability $2^{-(n-m-1)}$.*

Proof. We are checking the hypothesis that there are m consecutive zeroes separating the two ones. If $m = l$, then all checked equations are identities. Assume that $m \neq l$. Let k be the number of pair checked: $k = n - m - 1$. So the following equations are checked:

$$\begin{aligned}
a_i &= a_{i+(l-m)} \\
a_{i+1} &= a_{i+(l-m)+1} \\
&\dots \\
a_{i+k-1} &= a_{i+(l-m)+k-1}
\end{aligned}$$

Let us split those equations into separate “chains”:

$$\begin{aligned}
a_j &= a_{j+(l-m)} = a_{j+2(l-m)} = \dots \\
a_{j+1} &= a_{j+(l-m)+1} = a_{j+2(l-m)+1} = \dots \\
&\dots \\
a_{j+(l-m)-1} &= a_{j+2(l-m)-1} = a_{j+3(l-m)-1} = \dots
\end{aligned}$$

Let us consider a single chain $a_i = a_{i+(l-m)} = a_{i+2(l-m)} = \dots = a_{i+p(l-m)}$. According to our assumption a_i are random independent bits, so $\Pr\{a_i = 1\} = \frac{1}{2}$. Hence, the distribution of the vector $(a_i, a_{i+(l-m)}, a_{i+2(l-m)}, \dots, a_{i+p(l-m)})$ is also uniform. Then the number of its’ all possible states is 2^{p+1} , while only two of them yield a vector of all-equal elements. Then the probability of fulfilling the equations building the chain equals 2^{-p} .

Since all chains together contain k equations and the chains are disjoint, the events of fulfilling all chains’ equations are independent, which concludes the proof. $\square$

Lemma 1 shows that the probability of a false result grows rapidly with the assumed length of the gap between the ones. That is why we assume that the control sequence does not contain a block of more than *maxzeroes* zeroes. Of course, if this assumption is false, the algorithm will fail.

2.2 Full Attack

Now we assume that we still have all Z^k , but permuted according to an unknown, random permutation π . Such a situation can happen when we can stop the clocking for a couple of steps (no more than $n - 1$), but we cannot control the exact number of these steps. In such a scenario we would simply perform the experiments until we collect n different outputs, which would give us the complete set of outputs and no information about their ordering.

To perform an attack we first deduce the unknown permutation π , shuffle the outputs according to π^{-1} , and then perform the basic attack.

Assume for a moment that the control sequence C is of the form $00 \dots 011 \dots$. So if the Z^k sequences are in the correct order, the following equations hold:

$$z_2^1 = z_1^2, \quad z_2^2 = z_1^3, \quad \dots \quad z_2^{n-1} = z_1^{n-2}$$

(see Table 1); conversely, for Z^k and Z^l ($l \neq k + 1$) we have $z_2^k = z_1^l$ with probability $\frac{1}{2}$. Therefore we shall think of $Z^{\pi(i)}$'s as of vertices in a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{0, 1, 2, \dots, n - 1\}$ and there is an edge from vertex l to vertex k if and only if $z_2^k = z_1^l$. Similarly, if

$$C = 00 \dots 0 \underbrace{1 \dots 1}_m \dots, \quad (1)$$

then

$$\mathcal{E} = \left\{ (p, q) \in \mathcal{V} \times \mathcal{V} : p \neq q \wedge \bigwedge_{j=1}^{m-1} (z_{j+1}^p = z_j^q) \right\}. \quad (2)$$

Let $\{(v_0^i, v_1^i, \dots, v_{n-1}^i)\}_{i \in I}$ denote all Hamiltonian paths in $\mathcal{G}$. Then, for any given set of output sequences $Z^{\pi(j)}$ there exists $i \in I$ so that $v_j^i = \pi(j)$ holds for all $j \in \{0, 1, \dots, n - 1\}$. Of course, assumption (1) might be false, so it may be desirable to relax it: assume that among the first l control bits there are exactly k ones, and then there is continuous block of m ones. Then we could slightly modify the condition (2):

$$\mathcal{E} = \left\{ (p, q) \in \mathcal{V} \times \mathcal{V} : p \neq q \wedge \bigwedge_{j=1}^{m-1} (z_{j+1+k}^p = z_{j+k}^q) \right\}. \quad (3)$$

Of course, finding all Hamiltonian paths in a graph is a hard problem. However, in our restricted case it can be solved with a straightforward approach (since due to condition (3) the graph can be made almost arbitrarily sparse) and so we may use the following procedure:

- choose $m \in \{2, 3, 4, \dots\}$;

- for $k = 0, 1, 2, \dots$ construct the graph $\mathcal{G}$,
- find all possible Hamilton paths in $\mathcal{G}$; if none found, then k was incorrect; each path in $\mathcal{G}$ corresponds to some $\hat{\pi}(\cdot)$;
- for each $\hat{\pi}$ sort outputs $Z^{\pi(i)}$ according to $\hat{\pi}^{-1}$; if $\hat{\pi}$ was guessed right, then for all i holds $i = \hat{\pi}^{-1}(\pi(i))$, and so the output sequences are in the correct order;
- for every sorted set of output sequences perform the basic attack.

2.3 Simulation Results

The algorithm described was implemented and run on a low-end home PC, with an AMD Athlon XP1800+ CPU and 512MB of RAM. Their results for 100 runs for each choice of parameters are summarized by Table 2.

n	N	mz	c. p.	fails	avg. sol. count	avg. time	avg. time 90%
15	50	10	2	0	6086.89	4.5286	1.8920
15	50	10	4	11	4.56	0.0378	0.0047778
20	100	15	4	1	14.58	0.6402	0.071333
20	200	15	4	0	260.84	0.8431	0.10578
25	100	15	5	20	1.09	1.3391	0.048889
25	500	15	5	1	2.34	1.3790	0.10844
25	100	15	7	64	1.44	0.0178	0.0086667
25	500	15	7	12	1.69	0.0278	0.021667
30	100	15	6	44	1.00	5.4973	0.015778
30	500	15	6	1	1.00	1.0959	0.036111
35	100	20	6	39	1.00	0.7908	0.054111
35	500	20	6	1	1.00	1.0873	0.13156
35	100	20	8	86	1.00	0.0177	0.016667
35	500	20	8	44	1.04	0.0655	0.061444
35	100	25	6	40	2.02	0.5346	0.059444
35	500	25	6	1	1.31	9.7138	0.32267
35	100	25	8	83	1.12	0.0178	0.017444
35	500	25	8	37	2.13	0.3143	0.057444
40	100	25	7	69	1.00	0.0883	0.023000
40	1000	25	7	2	1.00	0.8059	0.099000
40	100	25	9	93	1.00	0.0253	0.023333
40	1000	25	9	35	1.02	0.1619	0.14811
45	100	25	8	78	1.00	0.0935	0.027778
45	1000	25	8	9	1.00	39.284	0.12200
45	100	25	9	86	1.00	0.0272	0.026111
45	1000	25	9	32	1.00	0.1899	0.17100
50	2000	30	10	32	1.00	9.2640	0.45800
50	2000	30	12	82	1.00	0.7974	0.78867
n	N	mz	c. p.	fails	avg. sol. count	avg. time	avg. time 90%

Table 2. Statistics of the attack; the columns have the following meaning: n – the number of output sequences, N – the length of output sequences, mz – the *maxzeroes* parameter, “c. p.” – the number of column pairs checked during construction of $\mathcal{G}$, “fails” – the number of fails (i.e. no solutions found), “avg. sol. count” – mean number of solutions (if found any), “avg. time” – mean calculations’ time (in seconds), “avg. time 90%” – mean calculations’ time after rejecting 10% of the worst cases.

3 An Attack with Random Faults

Now we assume that the attacker can disturb somehow the control sequence. It may be, for example, some additional cycles before the start of the system. Then he can observe only the output of the generator. The second assumption is that the procedure can be repeated with the same input sequence and with different faults on control sequence. Our goal is to obtain bits of the input sequence.

3.1 General Idea of the Attack

We assume a probabilistic model: we have an input sequence $A = a_1 a_2 a_3 \dots$ and a set of control sequences $C^{(i)} = c_1^{(i)} c_2^{(i)} c_3^{(i)} \dots$, $i = 1, 2, \dots, n$ where all $A, C^{(1)}, C^{(2)}, \dots, C^{(n)}$ are independent and truly random. For each $i = 1, 2, \dots, n$ a sequence $Z^{(i)}$ is produced from A and $C^{(i)}$ by the shrinking generator. The attacker knows only $Z^{(i)}$ for $i = 1, 2, \dots, n$ and tries to recover the sequence A .

Consider $z_1^{(i)}$, $i = 1, 2, \dots, n$. If most of them are equal to, say, 0, then it is natural to assume that a_1 also equals zero. This is concluded from the fact that about half of $z_1^{(i)}$, $i = 1, 2, \dots, n$, are a_1 . Similar considerations are to be carried on for a_2 , since about $\frac{1}{4}$ of $z_2^{(i)}$ are equal to a_2 , about $\frac{1}{4}$ of $z_2^{(i)}$ are equal to a_3 (control sequence is then of the form 011* or 101*) etc. Details of these fractions are given later. Of course situation is that clear if almost all of $z_j^{(i)}$ for a given j are equal to zero (or almost all are equal to one), but of course there are many intermediate situations which shall be discussed further.

Linear Equations Let us count how many times one of a_j is taken as one of $z_k^{(1)}, z_k^{(2)}, \dots, z_k^{(n)}$ and denote this number by $d(k, j)$. Now we can write the following system of equations:

$$\begin{cases} d(1, 1)a_1 + d(1, 2)a_2 + d(1, 3)a_3 + \dots = \sum_{j=1}^n z_1^{(j)}, \\ d(2, 2)a_2 + d(2, 3)a_3 + \dots = \sum_{j=1}^n z_2^{(j)}, \\ d(3, 3)a_3 + \dots = \sum_{j=1}^n z_3^{(j)}, \\ \dots\dots\dots \end{cases} \quad (4)$$

Of course the attacker does not know the numbers $d(k, j)$, but for large n he can approximate fractions $d(k, j)/n$ with appropriate probabilities. By $p_{k,j}$ denote the probability that j th element of a sequence A is taken as k th element of output sequence. More formally:

$$p_{k,j} = \Pr \left\{ \sum_{i=1}^{j-1} c_i = k - 1 \quad \wedge \quad c_j = 1 \right\} \quad (5)$$

where $c_1, c_2, \dots$ is a random control sequence. So, $d(k, j)$ can be approximated by $n \cdot p_{k,j}$.

It is interesting now to investigate the distribution of random variable X_k such that $X_k = j$ if and only if j th element of the sequence A is taken as

k th element of the output sequence Z . By definition $\Pr\{X_k = j\} = p_{k,j}$ and it follows from (5) that

$$p_{k,j} = \binom{j-1}{k-1} 2^{-j}, \quad \text{for } j \geq k, \quad \text{and} \quad p_{k,j} = 0, \quad \text{for } j < k. \quad (6)$$

Theorem 1.

$$EX_k = 2k \quad \text{and} \quad \text{Var } X_k = 2k.$$

Proof. By the definition of X_k and from (6) it is obvious that

$$\sum_{j=k}^{\infty} \binom{j-1}{k-1} 2^{-j} = 1, \quad \text{for } k \geq 1. \quad (7)$$

In order to prove the equation $EX_k = 2k$, we write

$$\begin{aligned} EX_k &= \sum_{j=k}^{\infty} j \cdot p_{k,j} = \sum_{j=k}^{\infty} j \binom{j-1}{k-1} 2^{-j} = \sum_{j=k}^{\infty} j \frac{(j-1)!}{(k-1)!(j-k)!} 2^{-j} \\ &= \sum_{j=k}^{\infty} k \frac{j!}{k!(j-k)!} 2^{-j} = k \sum_{j=k}^{\infty} \binom{j}{k} 2^{-j} = 2k \sum_{j=k+1}^{\infty} \binom{j-1}{(k+1)-1} 2^{-j} \end{aligned}$$

The last expression equals $2k$ due to equation (7). To prove that $\text{Var } X_k$, we first calculate EX_k^2 .

$$\begin{aligned} EX_k^2 &= \sum_{j=k}^{\infty} j^2 \cdot p_{k,j} = \sum_{j=k}^{\infty} j^2 \binom{j-1}{k-1} 2^{-j} \\ &= \sum_{j=k}^{\infty} j^2 \frac{(j-1)!}{(k-1)!(j-k)!} 2^{-j} = \sum_{j=k}^{\infty} k \cdot j \frac{j!}{k!(j-k)!} 2^{-j} \\ &= k \sum_{j=k}^{\infty} j \binom{j}{k} 2^{-j} = 2k \sum_{j=k+1}^{\infty} (j-1) \binom{j-1}{(k+1)-1} 2^{-j} \\ &= 2k \sum_{j=k+1}^{\infty} j \binom{j-1}{(k+1)-1} 2^{-j} - 2k \sum_{j=k+1}^{\infty} \binom{j-1}{(k+1)-1} 2^{-j} \\ &= 2k EX_{k+1} - 2k = 4k^2 + 2k. \end{aligned}$$

Now we obtain $\text{Var } X_k$ as $EX_k^2 - (EX_k)^2 = 2k$. $\square$

Knowing EX_k and $\text{Var } X_k$, we can approximate the number of variables that with high probability will be sufficient to build a finite version of system (4). Clearly, the exact probability $\Pr(X_k \leq x)$ is expressed by the formula

$$\Pr(X_k \leq x) = \sum_{j=k}^{\lfloor x \rfloor} \binom{j-1}{k-1} 2^{-j}. \quad (8)$$

The results obtained according to (8) are presented as Table 3.

k	1	10	20	30	40	50	100	1000
v	7	33	57	80	103	125	235	2106

Table 3. Maximum number v of variable that will suffice to build an equation number k (with probability 0.99), according to formula (8).

The formula (8) is rather complicated and that makes it hard to observe the overall tendency. So we proceed in order to find a compact formula which would approximate (8) and would enable us to conclude about properties of defined random variables.

Theorem 2. For random variables X_k defined earlier, for all integers $k \geq 1$ and for any real $p \in (0, 1)$ we have

$$\Pr\left(X_k \leq 2k \cdot \exp \sqrt{-2(\ln p)/k}\right) \geq 1 - p \quad (9)$$

Lemma 2. For random variables X_k defined above, and for all integers $k \geq 1$, $r \geq 1$ we have

$$EX_k^r \leq 2^r \frac{(k+r-1)!}{(k-1)!}. \quad (10)$$

Proof (of Lemma 2). If $r = 1$, then (10) is implied by Theorem 1. So assume that (10) is fulfilled for some r (and for all k) and we will prove it for $r+1$ and for all k . We pick an arbitrary k and write

$$\begin{aligned} EX_k^{r+1} &= \sum_{j=k}^{\infty} j^{r+1} \cdot p_{k,j} = \sum_{j=k}^{\infty} j^{r+1} \binom{j-1}{k-1} 2^{-j} \\ &= \sum_{j=k}^{\infty} j^{r+1} \frac{(j-1)!}{(k-1)!(j-k)!} 2^{-j} = \sum_{j=k}^{\infty} k \cdot j^r \frac{j!}{k!(j-k)!} 2^{-j} \\ &= k \sum_{j=k}^{\infty} j^r \binom{j}{k} 2^{-j} = 2k \sum_{j=k+1}^{\infty} (j-1)^r \binom{j-1}{(k+1)-1} 2^{-j} \\ &\leq 2k \sum_{j=k+1}^{\infty} j^r \binom{j-1}{(k+1)-1} 2^{-j} = 2k EX_{k+1}^r \\ &\leq 2k \cdot 2^r \frac{((k+1)+r-1)!}{((k+1)-1)!} = 2^{r+1} \frac{(k+(r+1)-1)!}{(k-1)!}. \quad \square \end{aligned}$$

Proof (of Theorem 2). Let $m > 1$ be some real number, we will approximate probability $\Pr(X_k > 2km)$. Pick any integer r . Since for all positive valued random variables Y we have $EY > w \cdot \Pr(Y > w)$, we can write

$$\Pr(X_k > 2km) = \Pr(X_k^r > 2^r k^r m^r) < \frac{EX_k^r}{2^r k^r m^r}.$$

Now due to Lemma 2 we have

$$\begin{aligned} \Pr(X_k > 2km) &\leq \frac{(k+r-1)!}{(k-1)!k^r m^r} = \frac{k(k+1)\dots(k+r-1)}{k^r} m^{-r} \\ &= \left(1 + \frac{1}{k}\right) \left(1 + \frac{2}{k}\right) \dots \left(1 + \frac{r-1}{k}\right) m^{-r} \\ &< \exp\left(\frac{1}{k}\right) \exp\left(\frac{2}{k}\right) \dots \exp\left(\frac{r-1}{k}\right) m^{-r} \\ &= \exp\left(\frac{r(r-1)}{2k}\right) m^{-r} < e^{\frac{r^2}{2k}} m^{-r}. \end{aligned}$$

Function $f(x) = e^{\frac{x^2}{2k}} m^{-x}$, $x \in \mathbf{R}_+$ reaches its absolute minimum for $x = k \ln m$, therefore we have

$$\Pr(X_k > 2km) < e^{\frac{(k \ln m)^2}{2k}} m^{-k \ln m} = e^{\frac{k(\ln m)^2}{2}} e^{-k(\ln m)^2} = e^{-\frac{k(\ln m)^2}{2}}.$$

Now let $p = e^{-k(\ln m)^2/2}$ then $m = \exp\left(\sqrt{\frac{-2 \ln p}{k}}\right)$ and for such m we have of course

$$\Pr\left(X_k > 2k \cdot \exp\left(\sqrt{\frac{-2 \ln p}{k}}\right)\right) < p.$$

Since there is a 1-to-1 correspondence between all $p \in (0, 1)$ and all reals $m > 1$, the theorem follows. $\square$

Theorem 2 shows that for a fixed small p , the ratio between the number of needed variables and expected value of X_k with high probability approaches 1 as k goes to ∞ . We can also see that for large enough k , going from equation number k to equation number $k + 1$ we will need only about 2 more variables.

3.2 The Algorithm and the Simulations

Input: n sequences $Z^{(1)}, Z^{(2)}, \dots, Z^{(n)}$, as described in 3.1, a number of equations w , a real number $p \in (0, 1)$. The closer p to zero, the better solution.

Output: $x_1, x_2, \dots, x_{w_v}$ – a candidate for the input sequence of the generator.

Method:

1. For each $k = 1, 2, \dots, w$, calculate the number v_k which guarantees inequality

$$\Pr(X_k \leq v_k) \geq 1 - p$$

2. Write the following approximation of the system (4):

[illegible]

3. Let $i := 1$, $v_0 := 0$ and $S_{\text{prev}} := \emptyset$.
 i numbers the equation considered in this step, S_{prev} is the set of solutions found after dealing with equations $1, 2, \dots, i - 1$.
4. Let $S := \emptyset$.
 S is the set solutions considered after regarding equation i .
5. Take some $s \in S_{\text{prev}}$. Let $S_{\text{prev}} := S_{\text{prev}} \setminus \{s\}$.
6. Consider all $2^{v_i - v_{i-1}}$ possible values of $x_{v_{i-1}+1}, \dots, x_{v_i}$. For each of them concatenate it with s , name the concatenation s' and let $S := S \cup \{s'\}$. If $|S| = M + 1$ remove the element of S with the highest rank. The choice of the rank function and M is a matter of discussion further on.
7. If $S_{\text{prev}} \neq \emptyset$, then go to Step 5.
8. $i := i + 1$.
9. If $i \leq w$, then put $S_{\text{prev}} := S$ and go to Step 4.
10. Take $s \in S$ with the lowest rank, and output v_w of its variables.

M , the maximum number of partial solutions kept while going from one equation to another, should be chosen carefully. If M is high (say, 2^{v_w}), then the algorithm would be equivalent to evaluating all possible guesses for v_w and therefore not practical even for $w = 20$ (see table 3). Too low values of M decrease chances to get a good solution. We have experimented with different values of M . Results for $M = 1000$ and $M = 10000$ are summarized in Table 4.

n	M	hits
100	1000	67.25% (13.16%)
100	10000	70.88% (12.28%)
1000	1000	78.00% (11.87%)
1000	10000	84.08% (11.14%)
10000	1000	93.29% (8.96%)
10000	10000	95.00% (8.78%)

Table 4. Average percentage of the first 20 bits of our solution that fit the real input sequence (standard deviation in parentheses) for different values of n and M . 120 experiments with $w = 40$ executed for each choice of parameters.

Now let us determine the rank function. Let the variables in a partial solution be $x_1, x_2, \dots, x_{v_i}$. Then

$$\text{rank}(s, i) = \sum_{m=1}^i \left(\frac{1}{n} \sum_{j=1}^n z_m^{(j)} - \sum_{j=m}^{v_m} p_{m,j} x_j \right)^2$$

Note that this formula lets us to use ranks calculated for equation $i - 1$, as our solutions are obtained from concatenation of partial solutions previously found with new guesses for a few new variables ($v_i - v_{i-1}$ is usually 2 or 3).

Although the algorithm outputs v_w bits of our guess of the input sequence, we have to stress that not all of the bits are of equal quality. The less is the number, the higher is the probability that the algorithm provides the right bit. Some experimental data is provided by Table 5.

n	f	hits
100	10	79.08% (17.70%)
100	20	70.88% (12.28%)
100	40	61.56% (8.59%)
100	60	58.51% (6.41%)
10000	10	100.00% (0.00%)
10000	20	95.00% (8.78%)
10000	40	77.96% (9.20%)
10000	60	69.50% (7.15%)

Table 5. Average percentage of correct values of bits (standard deviation in parentheses) for $M = 10000$ and $w = 40$. The results are computed for the first f bits of the algorithm output. Each row is a result of 120 trials.

One may ask how our assumption about independence of bits of control sequences stands in real situation. The Table 6 was calculated for control sequences generated by good quality LFSRs of length 32 with 4 taps. The faults are generated as one additional shift. One might expect much worse behavior, but the experiments show that the difference is insignificant.

n	M	hits
100	1000	65.50% (10.17%)
100	10000	67.79% (10.76%)
1000	1000	76.33% (12.63%)
1000	10000	79.83% (11.42%)
10000	1000	93.96% (8.49%)
10000	10000	95.63% (8.00%)

Table 6. The situation for faults generated by additional shifts in the control LFSR of length 32 and 4 taps. The table contains the average percentage of bits in the first 20 output bits that are reconstructed correctly (standard deviation in parentheses) for different values of n and M , and for $w = 40$. Each row is a result of 120 trials.

So far we can find most of few tens of bits, on average, of the input sequence. Finding sufficiently long sequence of bits is, in case of LFSR, equivalent to getting to know the whole sequences (it requires k bits in the case of known taps, and $2k$ bits in the case of unknown taps, where k is the number of bits of LFSR). If the attacker had some extreme power – like the possibility of obtaining, say, million different outputs for the same input sequence (and disturbed control sequences), then he could obtain more than 80% of 64 bits of the key. If one can have no more than 100 such sequences, then we expect to get about 60% of 64 bits. Observe (see the Table 5), that bits at the beginning of the sequence are more “reliable” than others, so if one wants to perform exhaustive check, the changing of bits should begin from “the right end” of bits being guessed. This way we will minimize searching, and in an average case most of guessing will be not necessary and the attack on 64 bits becomes realistic.

4 Conclusions and Open Problems

Our attacks can be easily adapted against the weaker generators [7, 1, 4] related to the shrinking generator. In those cases the attacks are even easier.

Our attacks require injecting specific faults and restarting the device with partially the same internal state. While injecting such faults is potentially possible (especially for the second attack) it may require some design faults (so that potentially vulnerable parts of the device were placed on external layers). It shows at least that careful examining of a chip design might be necessary.

Potentially, such attacks can be adapted against the other algorithms like stream ciphers, hash functions, and even some symmetric block ciphers (for instance injecting synchronization faults into DES key schedule seems to be beneficial due to its’ “shift-and-permute” design).

References

1. T. Beth, F. C. Piper, **The Stop-and-Go Generator**, EUROCRYPT’84, LNCS 209, pp. 88–92, Springer-Verlag, 1985.
2. D. Boneh, R. A. DeMillo, R. J. Lipton, **On the Importance of Checking Cryptographic Protocols for Faults**, EUROCRYPT ’97, LNCS 1233, pp. 37–51, Springer-Verlag, 1997.
3. D. Coppersmith, H. Krawczyk, Y. Mansour, **The Shrinking Generator**, CRYPTO’93, LNCS 773, pp. 22–39, Springer-Verlag, 1993.
4. W. Chambers, D. Gollmann, **Clock-Controlled Shift Registers: A Review**, IEEE J. Selected Areas Comm., 7(4): 525–533, May 1989.
5. E. Dawson, J. Dj. Golić, L. Simpson, **A Probabilistic Correlation Attack on the Shrinking Generator**, Information Security and Privacy – ACISP ’98, LNCS 1438, pp. 147–158, Springer-Verlag, 1998.
6. P. Ekdahl, Th. Johansson, W. Meier, **Predicting the Shrinking Generator with Fixed Connections**, EUROCRYPT 2003, LNCS 2656, pp. 330–344, Springer-Verlag, 2003.
7. P. R. Geffe, **How to Protect Data with Ciphers That Are Really Hard to Break**, Electronics, Jan. 4, pp. 99–101, 1973.
8. J. Dj. Golić, L. O’Connor, **Embedding and Probabilistic Correlation Attacks on Clock-Controlled Shift Registers**, EUROCRYPT ’94, LNCS 950, pp. 230–243, Springer-Verlag, 1995.
9. M. Gomułkiewicz, M. Kutyłowski, Th. H. Vierhaus, P. Wlaź, **Synchronization Fault Cryptanalysis for Breaking A5/1**, International Workshop on Efficient and Experimental Algorithms – WEA’05, LNCS 3503, pp. 415–427, Springer Verlag, 2005.
10. M. Krause, S. Lucks, E. Zenner, **Improved Cryptanalysis of the Self-Shrinking Generator**, Information Security and Privacy – ACISP 2001, LNCS 2119, pp. 21–35, Springer-Verlag, 2001.
11. W. Meier, O. Staffelbach, **The Self-shrinking Generator**, EUROCRYPT’94, LNCS 950, pp. 205–214, Springer-Verlag, 1995.
12. M. Mihaljevic, **A Faster Cryptanalysis of the Self-shrinking Generator**, Information Security and Privacy – ACISP ’96, LNCS 1172, pp. 182–188, Springer-Verlag, 1996.
13. T. R. N. Rao, Chung-Huang Yang, Kencheng Zeng, **An Improved Linear Syndrome Algorithm in Cryptanalysis With Applications**, CRYPTO’90, LNCS 537, pp. 34–47, Springer-Verlag, 1991.
14. E. Zenner, **On the Efficiency of the Clock Control Guessing Attack**, Information Security and Cryptology – ICISC 2002, LNCS 2587, pp. 200–212, Springer-Verlag, 2003.