On the Efficiency of the Clock Control Guessing Attack

Erik Zenner *

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

Abstract. Many bitstream generators are based on linear feedback shift registers. A widespread technique for the cryptanalysis of those generators is the linear consistency test (LCT). In this paper, we consider an application of the LCT in cryptanalysis of clock-controlled bitstream generators, called *clock control guessing*. We give a general and very simple method for estimating the efficiency of clock control guessing, yielding an upper bound on the effective key length of a whole group of bitstream generators. Finally, we apply the technique against a number of clock-controlled generators, such as the A5/1, alternating step generator, step1-step2 generator, cascade generator, and others.

1 Introduction

Pseudorandom bitstream generators are an important building block in modern cryptography. The design goal is to expand a short key into a long bitstream that is indistinguishable from a true random sequence by all computational means. In most cryptographic applications, the resulting pseudorandom bit sequence is added modulo 2 to the plaintext bit sequence.

In this paper, we consider the typical cryptanalytic situation for bitstream generators. The cryptanalyst is assumed to know the complete algorithmic description of the generator, with the exception of the inner state. Given a piece of bitstream, his goal is to reconstruct an initial state of the generator such that the generator's output is identical to the known bitstream.

Many bitstream generators are based on linear feedback shift registers (LFSR). An LFSR implements a linear recursion, transforming a short initial state into a long bit sequence. If the feedback recursion is chosen such that the corresponding feedback polynomial is primitive, the resulting sequence displays good statistical properties. In particular, all short substrings of length l occur with probability of almost exactly 2^{-l}. Throughout the rest of the paper, we assume that sequences generated by LFSR have this property.[1]

Nonetheless, a simple LFSR is an easy target for a cryptanalyst. Since the sequence generated by the LFSR is linear, recovering the initial state is only a

* Supported by the LGF Baden-Württemberg

[1] For more details on LFSR, refer to [7].

matter of solving a system of linear equations. Thus, LFSR must be employed in a more involved fashion, adding some non-linearity to the bitstream sequence.

One way of achieving this goal is clock-control. Clock-controlled generators do not clock all of their LFSR once per master clock, but rather use some irregular clocking instead. This way, the linearity of the resulting bit sequences is destroyed.

Organisation of the paper: In section 2, we will give a brief review of the linear consistency test (LCT) and its adaptive version as previously used in literature. In section 3, we discuss the technique of *clock control guessing*, an application of the LCT that was independently proposed in [14, 5, 11] for cryptanalysis of the A5/1 stream cipher. Now, we generalise the technique for use with a certain class of bitstream generators. Section 4 examines the efficiency of this technique, finding a surprisingly simple upper bound on the efficient key length of all involved generators. In section 5, we review the clock control guessing attack against A5/1 and give some experimental results. Section 6 considers a number of famous generators from literature. Finally, in section 7, some design recommendations and conclusions are given.

On notation: Throughout the paper, the length of the inner state of a generator will be denoted by L. The initial state $S(0)$ will sometimes be called "key" for simplicity. Each inner state $S(t)$ determines uniquely a clock control behaviour ξ_t (sometimes referred to as "clocking") that leads to the inner state $S(t+1)$. From the inner states $S(0), S(1), \ldots$, the generator derives a bitstream that is denoted by $y = (y_0, y_1, \ldots)$.

$$S(0) \xrightarrow{\xi_0} S(1) \xrightarrow{\xi_1} S(2) \xrightarrow{\xi_2} \ldots$$

When LFSR are used, they are denoted by A, B and C. LFSR A has length $|A|$ and generates a sequence $a = (a_0, a_1, \ldots)$; similarly for LFSR B and C.

Finally, by $\log(x)$ we denote the base-2 logarithm $\log_2(x)$.

2 LCT and adaptive bit guessing

Linear consistency tests: In [13], the *linear consistency test* (LCT) was formally introduced. In the meantime, it has become a rather widespread cryptographical technique, albeit the term "linear consistency test" is hardly used anymore. In short, the technique can be described as follows:

1. Choose a particularly useful subkey K_1 with $|K_1| < L$.
2. For all assignments κ for the subkey K_1:
3.   Derive the system of linear equations implied by κ.
4.   If the equation system is consistent:
5.   Output κ as subkey candidate.
6.   Else:
7.   Discard κ.

Usually, the equation system will be in at most L variables. Since in most practical applications, the linear equations can be read from a small precomputed table, each loop of the above algorithm takes $O(L^3)$ computational steps for solving a system of linear equations. Thus, the total running time of the algorithm is in the order of $O(L^3 \cdot 2^{|K_1|})$ computational steps.

Example: In literature, applications of this technique can often be found in the (much wider) category of divide-and-conquer attacks. As a simple example, consider the alternating step generator [8]. The generator consists of three LFSR C, A and B. For each clock t, the output c_t of LFSR C is determined. If $c_t = 0$, clock LFSR A, else clock LFSR B. Finally, add the current output bit of LFSR A and B (modulo 2) and append it to the bitstream.

Note that this generator can be attacked by a simple LCT attack. The cryptanalyst guesses the inner state of LFSR C. Now, he can compute the behaviour of the clock control and can form one equation of the form $a_i \oplus b_j = y_t$ per known output bit. Using the feedback recurrence, he can transform each such equation such that only variables from the starting state of LFSR A and B are being used. Finally, he checks the set of resulting equations for consistency.

Thus, the number of linear consistency tests equals $2^{|C|}$, taking less than $O(L^3)$ steps each (while the number of wrong key candidates should be negligibly small, see [13]). This might tempt a cipher designer to choose a large length for LFSR C at the cost of the length of LFSR A and B. However, in sections 3 and 4, we shall see that this is not a wise design decision.

Adaptive bit guessing: A variant of the plain LCT technique presented above can be denoted as *adaptive bit guessing*. It was used, e.g., by Golić in [4] in order to break the A5/1 stream cipher, or by Zenner, Krause, and Lucks in [15] for an attack against the self-shrinking generator.

The general idea is as follows. Instead of guessing all of the subkey in one go, the subkey bits are guessed one by one, allowing for instant verification of the linear equation system. This yields a backtracking attack on a clearly defined search tree. In many cases, this procedure has the advantage that (if an early contradiction occurs) a whole group of subkey candidates can be discarded at once, severely improving the running time. However, the running time of this attack is determined by the number of search tree nodes that are visited, and this number is often hard to determine in practice.

3 LCT and clock control guessing

Clock control guessing: In connection with clock-controlled bitstream generators, the LCT technique may be used in a slightly different way, yielding a very simple method of proving an upper bound on the running time. We consider clock control generators that have the following properties:

1. *The output bit depends on the inner state of the generator in some linear way.*

For each clock cycle t and each assignment to the output bit y_t, a linear equation q can be given such that the inner state $S(t)$ generates output bit y_t iff $S(t)$ is a solution to q.

2. *The behaviour of the clock control depends on the inner state of the generator in some linear way.*
   For each clock cycle t and each assignment to the clock control behaviour ξ_t, a set Q of linear equations can be given such that the inner state $S(t)$ generates the clock control value ξ_t iff $S(t)$ is a solution to Q.
3. *The number of possible behaviours of the internal clock is small.*

Given a generator that has properties 1-3, we can modify the adaptive bit guessing attack as follows. Instead of guessing individual bits, for each clock cycle $t = 0, 1, \ldots$, we guess the associated clocking ξ_t. We add all linear equations that follow from output bit y_t and clock control ξ_t to the linear equation system and check for consistency. The recursive method *clock_guess* in figure 1 gives the general idea of the attack.

clock_guess(*equ_system, clock_ctrl, t*)

```
 1. Build all linear equations from properties 1 and 2.
 2. Add equations to equ_system.
 3. If (LCT(equ_system)=false):
 4.      Start backtracking.
 5. t ← t + 1
 6. If (t = L):
 7.      Do exhaustive search on remaining key space.
 8.      Start backtracking.
 9. For all possible clockings ξ_t:
10.      clock_guess(equ_system, ξ_t, t).
```

Fig. 1. Recursive method *clock_guess*

Observation: Note that *clock_guess* implements a depth search on a tree, where each node of the tree contains a system of linear equations. Due to properties 1 and 2, all solutions to the equation system are keys that produce the bitstream $y_0, \ldots, y_{t-1}$. Consequently, steps 7-8 are only executed for keys that produce the bitstream $y_0, \ldots, y_{L-1}$. Since this property is only rarely met by random keys, the number of calls to steps 7-8 amongst all calls to *clock_guess* should be a very small integer. Thus, the average effort for steps 7-8 on a single call to *clock_guess* is negligible.

Considering that step 1 can be executed by a table lookup on a small precomputed table, it becomes obvious that the running time of one execution of *clock_guess* is dominated by steps 2 and 3. Here, the Gaussian algorithm for linear equation systems can be deployed, yielding an overall effort in $O(L^3)$ steps per call to *clock_guess*.

Alternating step generator, revisited: Applying the clock guessing attack against the alternating step generator, we would first guess c_0, then c_1, c_2 and so on[2]. Thus, we obtain two linear equations in each round (one for the clock control and one for the output bit) and wait for contradictions to occur. Note that - if LFSR A and B are much shorter than LFSR C - the first linear inconsistencies will occur long before the bit $c_{|C|}$ has been guessed, making clock control guessing much more efficient than a plain LCT attack.

4 On the efficiency of clock control guessing

Estimating the running time: As stated above, the running time of backtracking attacks is not easily determined. An important role plays the depth d of the nodes where the first inconsistent linear equation systems occur, and the probability of this event. For more involved bitstream generators, these values are not easily determined.

This is also true for the clock control guessing attack. A precise estimate of the running time (i.e., the number of calls to *clock_guess*) is not possible without paying close attention to the details of the cipher considered. The length of the registers, the sparseness of the feedback polynomials, the positions of the output and clock control bits and the choice of the output and clock control function all determine the efficiency of the attack.

We can, however, prove a general upper bound for the size of the search tree considered. In order to do this, we assume that the generator meets the following condition:

4. *The number of initial states $S(0)$ that are consistent with the first d output bit ($d \leq L$) is approximately 2^{L-d}.*

Note that this condition is met by all properly designed bitstream generators, since otherwise, correlation attacks are easily implemented. Now we can estimate the maximum width of the search tree, using an elegant technique proposed by Krause in [9]. First, we make some simple observations.

Observation 1: Consider a node v in the search tree at depth d. Such a node is reached by a sequence $c_0, c_1, \ldots, c_{d-1}$ of guesses for the clock control behaviour. It contains a system V of linear equations derived on the path from the root to the node by using properties 1 and 2 of the generator. The set of solutions to V has the following properties:

a) All solutions to V produce the clock control sequence $c_0, c_1, \ldots, c_{d-1}$.
b) All solutions to V produce the bitstream sequence $y_0, y_1, \ldots, y_{d-1}$.
c) If V is consistent, there is at least one solution to V.

[2] Note that for the alternating step generator, the clock control guessing attack is identical to the adaptive bit guessing attack.

We say that the node v *represents* all inner states that are solutions to V, and that v is consistent if V is consistent. As a consequence of property a, no two nodes at depth d represent the same inner state, since different nodes imply different behaviours of the clock control. On the other hand, no node v represents an inner state that is inconsistent with the output bits $y_0, \ldots, y_{d-1}$. From property 4 of the generator, we know that there are approximately 2^{L-d} solutions in all of the nodes. Since by property c, there are no empty consistent nodes, there can be at most 2^{L-d} consistent nodes at depth d. For low values of d, however, the number of consistent nodes is going to be a lot smaller since each node represents a huge number of inner states.

Observation 2: On the other hand, the number of nodes in the tree at depth d can never be larger than k^d, where k is the number of possible behaviours of the clock control. For small values of d, this estimate will usually be exact, while for larger values of d, the actual tree contains a lot less nodes than indicated by this number.

Width of the search tree: Observe that the function 2^{L-d} is constantly decreasing in d, while k^d is constantly increasing. Since the number of consistent nodes in the tree is indeed upper bounded by both of these functions, the maximum number of nodes at a given depth is upper bounded by $\min\{2^{L-d}, k^d\}$. If we write $k^d = 2^{\log(k)\cdot d}$ for convenience, the maximum number of nodes must be smaller than 2^w with $w = L - d$, yielding

$$\begin{aligned} 2^w &= 2^{\log(k)\cdot(L-w)} \\ w &= \log(k) \cdot (L - w) \\ w &= \frac{\log(k)}{\log(k)+1} L \end{aligned}$$

Thus, the number of consistent nodes in the widest part of the search tree can not exceed $2^{\lambda L}$ with $\lambda = \frac{\log(k)}{\log(k)+1}$. Note that this is not an asymptotical result; it is perfectly valid to use concrete values for k and L and to calculate the upper bound.

Total running time: Now that we have obtained an upper bound on the width of the search tree, the total running time is easily determined. Observing that

- there are at most two layers with width 2^w, that
- all layers above those two have at most 2^w consistent nodes amongst them, and that
- all layers below those two have at most 2^w consistent nodes amongst them,

we see that the tree has at most $4 \cdot 2^w$ consistent nodes. Observing further that there must be less than k non-consistent nodes for each consistent node, we obtain a maximum of $4\cdot(k+1)\cdot 2^w \in O(2^w)$ recursive calls to method *clock_guess*. Thus, remembering our observation from section 3, the overall running time must be in the order of $O(L^3 \cdot 2^{\lambda L})$ with $\lambda = \frac{\log(k)}{\log(k)+1}$.

Alternating step generator, concluded: Let us use our new result on the alternating step generator. There are only two options for the clock control, yielding $\log(k) = \log(2) = 1$ and thus $w = L/2$. Consequently, quite independent of the choice of the individual parameters, any implementation of the alternating step generator can be broken by a clock control guessing attack in $O(L^3 \cdot 2^{0.5L})$ steps, yielding an absolute upper bound of $0.5L$ bit on the efficient key size of this kind of generator. In particular, increasing the length of LFSR C while decreasing the lengths of LFSR B and C (as proposed in section 2) can not possibly increase security beyond this point. Also note that depending on the choice of the individual parameters, the attack may even be much more efficient.

5 Application: Attacking A5/1

Description of the cipher: A5/1 is the encryption algorithm used by the GSM standard for mobile phones; it was described in [2]. The core building block is a bitstream generator, consisting of three LFSR with a total length of 64 bit. First, the output is generated as the sum (mod 2) of the least significant bits of the three registers. Then the registers are clocked in a stop-and-go fashion according to the following rule:

- Each register delivers one bit to the clock control. The position of the clock control tap is fixed for each register.
- A register is clocked iff its clock control bit agrees with the majority of all clock control bits.

Clock control guessing: As mentioned before, the clock control guessing attack on A5/1 was discussed earlier by Zenner [14], Golić [5], and Pornin and Stern [11]. First observe that the A5/1 generator produces 1 output bit per master clock cycle, and that there are 4 different behaviours of the clock control. Let u_1, u_2 and u_3 denote the contents of the clock control bits for a given clock cycle. Table 1 gives the dependency between u_1, u_2, u_3 and the behaviour C of the clock control. Note that equivalent linear equations are easily constructed. Thus, we

C	Equation
(011)	$u_1 \neq u_2 = u_3$
(101)	$u_1 \neq u_2 \neq u_3$
(110)	$u_1 = u_2 \neq u_3$
(111)	$u_1 = u_2 = u_3$

Table 1. Clock control and linear equations

see that the A5/1 algorithm meets all prerequisites for a successful clock control guessing attack. We simply guess the behaviour of the clock control for each output bit, derive the linear equations and check for consistency.

Upper bounding the running time: Applying our estimate technique to the A5/1, we have to observe two facts:

1. The initial state is generated in such a way that only $\frac{5}{8} \cdot 2^{64}$ states are in fact possible. The impossible states can be excluded by a number of simple linear equations (for details, see [4]). Thus, the efficient key length of the inner state is only $64 + \log(\frac{5}{8}) \approx 63.32$ bit.
2. Furthermore, the first output bit is not yet dependent on the clock control. Thus, the efficient key length of the inner state prior to any clock control guessing is further reduced by 1 bit, yielding $L \approx 62.32$.

For each master clock cycle, 4 possible behaviours of the clock control are possible. Thus, $k = 4$ and $\log(k) = 2$. Using the estimate from section 4, we conclude that the search tree has a maximum width of $2^{(2/3)\cdot 62.32} \approx 2^{41.547}$ nodes.

This result coincides with the maximum number of end nodes as given by Golić in [5], derived from a more involved analysis. Also note that in the same work, the average number of end nodes was estimated to be $2^{40.1}$, as was to be expected. By paying close attention to important details of the generator such as the position of the feedback taps or the length of the registers, an estimate for the tree size can be derived that in most cases will be lower than the general upper bound. Nonetheless, this upper bound gives a first indication of a cipher's strength by ruling out some weak ciphers without further effort.

Test run on a small version: In order to demonstrate the difference between the proven upper bound and the actual running time, we have implemented a 40-bit version of the A5/1, featuring the details given in table 2.

LFSR	length	feedback polynomial	clock control tap
A	11	$x^{11} + x^2 + 1$	a_6 (in $a_0, \ldots, a_{10}$)
B	14	$x^{14} + x^5 + x^3 + x + 1$	b_7 (in $b_0, \ldots, b_{13}$)
C	15	$x^{15} + x^{12} + x^4 + x^2 + 1$	c_8 (in $c_0, \ldots, c_{14}$)

Table 2. 40-bit version of the A5/1 generator

Again, we observe that the first output bit is not yet dependent on the clock control, yielding 2^{39} candidates for the initial state or an efficient key length of $L = 39$ bit.[3] Thus, we would expect the bounding functions to be 4^d and 2^{39-d}, yielding a maximum search tree width of 2^{26}.

An overall of 120 experiments was conducted, and the results are shown in figure 2. The figure shows the average width of the search trees that were found in the experiments. It also gives the bounding functions 4^d and 2^{39-d} for convenience. The following observations can be made:

[3] For simplicity's sake, we ignore the fact that only $\frac{5}{8} \cdot 2^{40}$ inner states are actually possible.

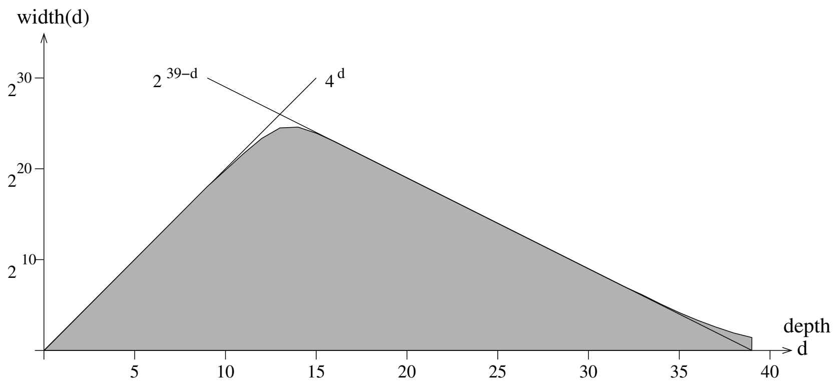

Fig. 2. Width of search tree for small A5/1 generator

- The actual tree width at depth d matches the predicted value of $\min(4^d, 2^{39-d})$ surprisingly well.
- In the widest part of the tree ($d = 14$), the actual number of nodes is smaller than the predicted upper bound, which was to be expected.
- In the lowest part of the tree ($d > 34$), the actual number of nodes is larger than predicted by the function 2^{39-d}. This is due to the fact that for the A5/1 generator, there is a chance that several inner states map onto the same output sequence, i.e., assumption 4 does not hold for high values of d. This, however, does not affect the performance of the algorithm, since the running time is almost exclusively determined by the widest part of the tree.

In our experiments, we found an average of 1.758 inner states that produce the same output. Judging from the empirical data as given in table 3, it seems that the probability of a bitstream (generated from a random seed) having z generating keys is approximately 2^{-z} for small values of z. Whether or not this assumption is correct and whether or not it also holds for the full version of A5/1 remains an open problem.

equivalent keys	1	2	3	4	5	6	7
frequency	64	33	17	2	3	-	1

Table 3. Frequency of equivalent keys

6 Other Generators

In this section, we will review some generators from literature, pointing out some dos and don'ts when using the above attack and the associated technique for upper bounding the efficient key length.

Stop-and-go generator: The stop-and-go generator [1] consists of two LFSR C and A, where the output bit is taken as the least significant bit of LFSR A. While LFSR C is clocked regularly and outputs $c_1, c_2, \ldots$, LFSR A is clocked iff $c_t = 1$. As a consequence, the output sequence y has a probability of 3/4 that the condition $y_t = y_{t-1}$ holds. Thus, certain output sequence prefixes are much more likely than others, contradicting property 4. Thus, even though the clock control guessing attack can be implemented against the stop-and-go generator, the estimate can not be used without further thought.

Step1-step2 generator: The step1-step2 generator [6] modifies the stop-and-go generator in that depending on bit c_t, the LFSR A is stepped once ($c_t = 0$) or twice ($c_t = 1$). In this case, the resulting bit sequence does not display the anomaly of the stop-and-go generator and meets property 4. Since the behaviour of the clock control can be described as for the alternating step generator and since there are only 2 possible behaviours of the clock control, we obtain an upper bound of $2^{0.5L}$ for the efficient key length of the step1-step2 generator, independent of the individual parameters.

Cascading generator: Many clock-controlled generators (such as stop-and-go generator and step1-step2 generator) can be generalised into a cascade by using more than just 2 LFSR. Typically, the output bit of LFSR i controls the clocking of LFSR $i+1$ and is also added to the output of LFSR $i+1$ modulo 2. In [6], Gollmann and Chambers describe some possible constructions for cascade generators obtaining good statistical bitstream properties.

If the basic clock-control mechanism (e.g., stop-and-go) meets conditions 1-3, then the associated cascade generator can be attacked using clock control guessing. After making sure that the resulting bitstream meets assumption 4, we can use the above technique to derive an upper bound on the effective key length. Given a cascade with s LFSR and a total bit length of L, we see that there are $k = 2^{s-1}$ possible behaviours for the clock control. Thus, we have $\log(k) = s - 1$ and the efficient key length is at most $\frac{s-1}{s}L$.

Note that this is not identical to the naïve divide-and-conquer attack of guessing the contents of the uppermost $s-1$ registers and deriving the content of the lowest LFSR from the bitstream. This naïve attack has computational cost in the order of $O(2^{L-l})$, where l is the length of the lowest LFSR. If $l < \frac{L}{s}$, the clock control guessing attack will usually be more efficient than the simple divide-and-conquer attack.

Shrinking generator: The shrinking generator was proposed in [3]. It consists of two LFSR C and A that are clocked simultaneously. For each clock t , if the output bit c_t of LFSR C equals 1, the output bit a_t of LFSR A is used as output bit. Otherwise, a_t is discarded.

Note that this generator can be viewed as a clock-controlled generator, where register A is clocked once with probability $1/2$, twice with probability $1/4$ a.s.o. before producing one bit of output. Thus, the number of possible clock control behaviours is rather large (up to $|C|$ different possibilities), the property 3 is violated and the attack is not applicable in a straightforward manner. In this case, the adaptive bit guessing attack seems to obtain better results[4].

7 Conclusions

We have presented the cryptanalytic technique of clock control guessing which is applicable against a large number of clock-controlled bitstream generators. We have also given a general technique for upper bounding the efficiency of our attack, yielding an efficient key length of at most $\frac{\log(k)}{\log(k+1)}L$ bit, where k is the number of possible behaviours for the clock control.

Most clock-controlled generators proposed in the literature have rather simplistic clock control rules, often yielding $k = 2$ and thus cutting the efficient key length down to $L/2$ even without more detailed analysis. If this is not acceptable, any of the following design changes increases resistance against our attack:

- Increase the number of possible behaviours for the clock control. This way, the search tree expands rather rapidly, making the search more difficult.
- Choose a non-linear function for the clock control.
- Choose a non-linear function for the keybit extraction.

A generic example of a clock-controlled bitstream generator that can be designed to follow all of those design criteria is the LILI generator [12]. The generator consists of two LFSR C and A, where C determines the clock control and A the output. The clock control c_t is determined from the inner state of LFSR C by a bijective function $f_c : \{0,1\}^m \to \{1,\ldots,2^m\}$, and the output bit y_t is computed from the inner state of LFSR A using a Boolean function $f_d : \{0,1\}^n \to \{0,1\}$. If the values m and n are chosen large enough and if the functions f_c and f_d are non-linear, the generator should be safe from clock control guessing attacks[5].

References

1. T. Beth and F. Piper. The stop-and-go generator. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology - Eurocrypt '84*, volume 209 of *LNCS*, pages 88–92. Springer, 1985.

[4] The same observation holds for the self-shrinking generator, presented in [10].

[5] The mapping $f_c(x_1,\ldots,x_k) = 1 + x_1 + 2x_2 + \ldots + 2^{k-1}x_k$ that was proposed by the authors is easily modelled using linear equations. This should not be a problem, as long as the other design criteria are met. For paranoia's sake, however, a non-linear permutation might be considered instead.

2. M. Briceno, I. Goldberg, and D. Wagner. A pedagogical implementation of A5/1. `http://www.scard.org/gsm/a51.html`.
3. D. Coppersmith, H. Krawczyk, and Y. Mansour. The shrinking generator. In D.R. Stinson, editor, *Advances in Cryptology - Eurocrypto '93*, volume 773 of *LNCS*, pages 22–39, Berlin, 1993. Springer.
4. 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.
5. J.D. Golić. Cryptanalysis of three mutually clock-controlled stop/go shift registers. *IEEE Trans. Inf. Theory*, 46(3):1081–1090, May 2000.
6. D. Gollmann and W. Chambers. Clock-controlled shift registers: A review. *IEEE J. Selected Areas Comm.*, 7(4):525–533, May 1989.
7. S. Golomb. *Shift Register Sequences.* Aegean Park Press, Laguna Hills (CA), revised edition, 1982.
8. C. Günther. Alternating step generators controlled by de Bruijn sequences. In D. Chaum and W. Price, editors, *Advances in Cryptology - Eurocrypt '87*, volume 304 of *LNCS*, pages 88–92. Springer, 1988.
9. M. Krause. BDD-based cryptanalysis of keystream generators. In L. Knudsen, editor, *Advances in Cryptology - Eurocrypt '02*, LNCS. Springer, 2002.
10. 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.
11. T. Pornin and J. Stern. Software-hardware trade-offs: Application to A5/1 cryptanalysis. In Ç. Koç and C. Paar, editors, *Proc. CHES 2000*, volume 1965 of *LNCS*, pages 318–327. Springer, 2000.
12. L. Simpson, E. Dawson, J. Golič, and W. Millan. LILI keystream generator. In D. Stinson and S. Tavares, editors, *Proc. SAC 2000*, volume 2012 of *LNCS*, pages 248–261. Springer, 2001.
13. K. Zeng, C. Yang, and Y. Rao. On the linear consistency test (LCT) in cryptanalysis with applications. In G. Brassard, editor, *Advances in Cryptology - Crypto '89*, volume 435 of *LNCS*, pages 164–174. Springer, 1990.
14. E. Zenner. Kryptographische Protokolle im GSM-Standard - Beschreibung und Kryptanalyse. Master's thesis, University of Mannheim, 1999.
15. E. Zenner, M. Krause, and S. Lucks. Improved cryptanalysis of the self-shrinking generator. In V. Varadharajan and Y. Mu, editors, *Proc. ACISP '01*, volume 2119 of *LNCS*, pages 21–35. Springer, 2001.