A New Simple Technique to Attack Filter Generators and Related Ciphers

Håkan Englund and Thomas Johansson

Dept. of Information Techonolgy, Lund University,
P.O. Box 118, 221 00 Lund, Sweden

Abstract. This paper presents a new simple distinguishing attack that can be applied on stream ciphers constructed from filter generators or similar structures. We demonstrate the effectiveness by describing key recovery attacks on the stream cipher LILI-128. One attack on LILI-128 requires 2^{47} bits of keystream and a computational complexity of roughly 2^{53}. This is a significant improvement compared to other known attacks.

1 Introduction

Much work has been put into trying to understand the security of stream ciphers. Stream ciphers can be made very efficient in software and in hardware, but their security has not been as widely studied as for example block ciphers. In this paper we investigate filter generators, a linear feedback shift register (LFSR) from which the output is filtered by a nonlinear filter function. This output is added modulo two to the plaintext. See for example [18] for more details on filter generators.

Several different kinds of attacks can be considered on stream ciphers. We usually consider the plaintext to be known, i.e. the keystream is known and we try to recover the key. A popular technique is to exploit some correlation in the keystream. This idea was introduced by Siegenthaler [23] in 1984, a consequence of this attack is that designers of nonlinear functions must use functions with high nonlinearity. This attack was later followed by the fast correlation attack by Meier and Staffelbach [17]. Since then many improvements have been introduced on this topic, see [1, 2, 14, 13, 15]. In a fast correlation attack one first try to find a low weight parity check polynomial of the LFSR and then apply some iterative decoding procedure.

Algebraic attacks have received much interest lately. These attacks try to reduce the key recovery problem to the problem of solving a large system of algebraic equations [6, 5].

Another class of key recovery attacks on filter generators was proposed by Golić, the so-called inversion attacks, see [10, 11, 12]. In an inversion attack one tries to "invert" the nonlinear function and recover the initial state.

A distinguishing attack is a different type of attack. Here we try to distinguish the output of the cipher from a truly random source. In some specific cases these

H. Handschuh and A. Hasan (Eds.): SAC 2004, LNCS 3357, pp. 39–53, 2005.

attacks can be used to create a key recovery attack. Distinguishing attacks have received a lot of attention recently, see for example [8, 4].

In this paper we present a very simple distinguishing attack that can be applied on stream ciphers using a filter generator or a similar structure as a part of the cipher.

Recently, Leveiller et. al. [16] proposed methods involving iterative decoding and the use of vectors instead of the binary symmetric channel. We use a similar idea, but much simpler in its form and more powerful in its performance, to mount a distinguishing attack. In the basic algorithm we first find a low weight multiple of the LFSR. We then consider the entries of the parity check equation as a vector. Such vectors, regarded as random variables, are non-uniformly distributed due to the parity check, and this is the key observation that we use to perform a distinguishing attack. This allows us to detect statistical deviations in the output sequence, creating the distinguishing attack. We can also present ideas on how to improve the performance by using slightly more complex algorithms.

In order to demonstrate the effectiveness of the proposed ideas, we apply them on a recently proposed cipher called LILI-128. The attack is a *key recovery attack*. LILI-128 has one LFSR controlling the clock of another LFSR. Our approach is to guess the first 39 bits of the key, those bits that are used in the LFSR that controls the clocking. If our guess is correct we will be able to detect some bias in the output sequence through the proposed distinguishing attacks. The complexity for one of the proposed attacks is roughly 2^{53} binary operations and it needs about 2^{47} keystream bits, a significant improvement compared to other known attacks.

The paper is organized as follows. In Section 2 we give a basic description of filter generators. In Section 3 we present some theory on hypothesis testing. After this we describe our new distinguishing attack in Section 4, here we also present some ideas on how to improve this distinguishing attack. In Section 5 we turn this attack into a key recovery attack on LILI-128. Finally in Section 6 some future work and conclusions are discussed.

2 Preliminaries

In this paper we consider binary stream ciphers where the output from a LFSR is filtered by a nonlinear function. The keystream generator is divided into two parts, one linear, i.e., the LFSR, and one nonlinear function. LFSRs are known to produce long pseudo-random data sequences and can be made very efficient in both hardware and software. Usually the feedback polynomial of the shift register is primitive and the LFSR sequence will have maximum period. Since the initial state of an LFSR is very simple to recreate from the output stream we need to destroy the linearity in the keystream. This is the purpose of the nonlinear function. Much work on nonlinear functions has been done, see for example [21].

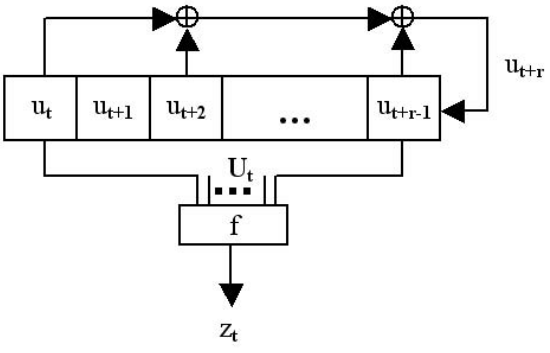

Fig. 1. Description of a filter generator

A filter generator can be described as follows. Let $u_t, u_{t+1}, \ldots$ denote the output sequence from a length r LFSR with feedback polynomial $g(x)$. Let f denote the nonlinear function as can be seen in Figure 1. At each time t this function takes d input values from the LFSR register and produces on output bit z_t. The variables used as inputs to f at time t are the entries in the vector $\mathbf{U}_t = (u_{t+t_1}, u_{t+t_2}, \ldots, u_{t+t_d})^T$. We denote the output sequence from the Boolean function, i.e., the keystream, by $z_t, z_{t+1}, \ldots$. We thus have $z_t = f(\mathbf{U}_t)$. This is modeled in Figure 1.

3 Hypothesis Testing

In a distinguishing attack we try to decide whether the data origins from the considered cipher or from a random source. To make this decision we use hypothesis testing. The problem stated above can be reformulated. We have two hypotheses, where H_0 denotes the hypothesis that the observed data comes from our cipher and H_1 that the data origins from a random source. We will now shortly explain how the decision is made and how we can calculate the number of samples we need to make a correct decision. For a more thorough description of hypothesis testing, see [7].

Assume that we have a sequence of m independent and identically distributed (i.i.d.) random variables $X_1, X_2, \ldots, X_m$ taken from the alphabet $\mathcal{X}$. The distribution of the random variables, X_i, are denoted $P(x) = \Pr(X_i = x)$, $1 \leq i \leq m$, where $x_1, x_2, \ldots, x_m$ denotes observed values. If we denote the distribution of X_i under hypothesis H_0 with P_0 and the uniform distribution by P_1, we can write our hypothesizes as $H_0 : P = P_0$ and $H_1 : P = P_1$. To perform the actual hypothesis test we use the Neyman-Pearson lemma.

Lemma 1. *(Neyman-Pearson lemma) Let $X_1, X_2, \ldots, X_m$ be drawn i.i.d. according to mass function P. Consider the decision problem corresponding to the hypotheses $P = P_0$ vs. $P = P_1$. For $T \geq 0$ define a region.*

$$\mathcal{A}_m(T) = \left\{ (x_1, x_2, \ldots, x_m) : \frac{P_0(x_1, x_2, \ldots, x_m)}{P_1(x_1, x_2, \ldots, x_m)} > T \right\}.$$

Let $\alpha = P_0(\mathcal{A}_m^c(T))$ and $\beta = P_1(\mathcal{A}_m(T))$ be the error probabilities corresponding to the decision region $\mathcal{A}_m$, ($\mathcal{A}_m^c$ denotes the complement of the region $\mathcal{A}_m$). Let $\mathcal{B}_m$ be any other decision region with associated error probabilities α^* and β^*. If $\alpha^* \leq \alpha$, then $\beta^* \geq \beta$.

The region $\mathcal{A}_m(T)$ minimizes α and β. In our case we set α and β to be equal and hence $T = 1$. As all x_n are assumed independent we can rewrite the Newman-Pearson as a log-likelihood test,

$$I \;=\; \sum_{n=1}^{m} \left(\log_2 \frac{P_0(x_n)}{P_1(x_n)} \right) > 0 \;? \tag{1}$$

We also need to know how many keystream bits we need to observe in order to make a correct decision. In [4] the *statistical distance* is used. The statistical distance, denoted ϵ, between two distributions P_0, P_1 defined over the finite alphabet $\mathcal{X}$, is defined as

$$\epsilon = |P_0 - P_1| = \sum_{x \in \mathcal{X}} |P_0(x) - P_1(x)|, \tag{2}$$

where x is an element of $\mathcal{X}$. Since $0 \leq \epsilon \leq 2$ we use the more natural $\varepsilon = \epsilon/2$. If the distributions are smooth, the number of variables N we need to observe is $N \approx 1/\varepsilon^2$, see [4]. Note that the error probabilities are decreasing exponentially with N.

4 Description of the New Attack

In this section we will give a description of the different steps of our attack. If the feedback polynomial of the LFSR is of low weight from the beginning, we can apply our attack directly. Usually this is not the case, and our first step is then to try to find a low weight multiple of the feedback polynomial.

4.1 Finding a Low Weight Multiple

There exist many methods for finding low weight multiples (of weight w) of a feedback polynomial $g(x)$. Because the degree of the multiple gives a lower bound of the number of samples we need to observe, we wish to minimize this degree. In [9] it is stated that the critical degree when the polynomial multiples of weight w starts to appear is $(w-1)!^{1/(w-1)} 2^{r/(w-1)}$, where r is the degree of the original feedback polynomial. In [9] an algorithm to find multiples is described. First one calculates the residues $x^i \bmod g(x)$, then one computes the residues $x^{i_1} + \ldots x^{i_k} \bmod g(x)$ for all $\binom{n}{k}$ combinations $1 \leq i_1 \leq \ldots \leq i_k \leq n$, with n

being the maximum degree of the multiples. Use fast sorting to find all of the zero and one matches of the residues from the second step. The complexity of this algorithm is approximately $O(S \log S)$ with $S = \frac{(2k)!^{1/2}}{k!} 2^{r/2}$ for odd multiples of weight $w = 2k+1$, and $S = \frac{(2k-1)!^{k/(2k-1)}}{k!} 2^{rk/(2k-1)}$ for even multiples of weight $w = 2k$.

Wagner [24] presented a generalization of the birthday problem, i.e., given k lists of r-bit values, find a way to choose one element from each list, so that these k values XOR to zero. This algorithm finds a multiple of weight $w = k+1$ with lower complexity, $k \cdot 2^{r/(1+\lfloor \log k \rfloor)}$, than [9] but with higher degrees, $2^{r/(1+\lfloor \log k \rfloor)}$, on the multiples. Since the number of samples is of high concern to us we have chosen to work with the method described in [9]. Continuing, we now assume that the LFSR sequence is described by a low weight recursion.

4.2 Building a Distinguisher

The technique we use for building a distinguisher is inspired by the work in [16]. However in [16] the authors describe a key recovery attack and use iterative decoding methods, etc. We construct instead a very simple distinguisher. A usual description of a stream cipher is to model it as a binary symmetric channel (BSC), using linear approximations. But we proceed differently. Instead we write the terms in the weight w parity check equation as a length w vector. This way we use our knowledge of the nonlinear function better than in the BSC model. Assume that we have a LFSR of weight w with the parity check equation

$$u_t + u_{t+\tau_1} + \ldots + u_{t+\tau_{w-1}} = 0. \tag{3}$$

We write the terms in this relation as a vector, and by noticing that $u_{t+\tau_{w-1}}$ is fully determined by the sum of the other components we get ($\tau_0 = 0$),

$$(u_t, u_{t+\tau_1}, \ldots, u_{t+\tau_{w-1}}) = (u_t, u_{t+\tau_1}, \ldots, \sum_{i=0}^{w-2} u_{t+\tau_i}). \tag{4}$$

From the LFSR, d different positions are taken as input to the nonlinear function f. For each of these positions, where $t_1, t_2, \ldots, t_d$ denotes its position relative to time t, we can write a vector similar to (4). If we consider the following matrix,

$$A_t = \begin{pmatrix} u_{t+t_1} & u_{t+t_1+\tau_1} & \cdots & \sum_{i=0}^{w-2} u_{t+t_1+\tau_i} \\ u_{t+t_2} & u_{t+t_2+\tau_1} & \cdots & \sum_{i=0}^{w-2} u_{t+t_2+\tau_i} \\ \vdots & \vdots & & \vdots \\ u_{t+t_d} & u_{t+t_d+\tau_1} & \cdots & \sum_{i=0}^{w-2} u_{t+t_d+\tau_i} \end{pmatrix},$$

then by writing $\mathbf{U}_{t+\tau_l} = (u_{t+t_1+\tau_l}, u_{t+t_2+\tau_l}, \ldots, u_{t+t_d+\tau_l})^T$ we get

$$A_t = (\mathbf{U}_t, \mathbf{U}_{t+\tau_1}, \ldots, \sum_{i=0}^{w-2} \mathbf{U}_{t+\tau_i}). \tag{5}$$

In the attack we will not have access to the LFSR output, instead we have access to the output bits from the nonlinear function f. The output values $(z_t, z_{t+\tau_1}, \ldots, z_{t+\tau_{w-1}})$, denoted by $\mathbf{Z_t}$, can be described as

$$\mathbf{Z_t} = (z_t, z_{t+\tau_1}, \ldots, z_{t+\tau_{w-1}}) = (f(\mathbf{U}_t), f(\mathbf{U}_{t+\tau_1}), \ldots, f(\sum_{i=0}^{w-2} \mathbf{U}_{t+\tau_i})). \quad (6)$$

As we run through $\mathbf{U}_t, \mathbf{U}_{t+\tau_1}, \ldots, \mathbf{U}_{t+\tau_{w-1}}$ in a nonuniform manner (not all values of $\mathbf{U_t}, \mathbf{U_{t+\tau_1}}, \ldots, \mathbf{U_{t+\tau_{w-1}}}$ are possible), we will (in general) generate a nonuniform distribution of $(z_t, z_{t+\tau_1}, \ldots, z_{t+\tau_{w-1}})$. In [16] it is shown that the distribution of these vectors only depends on the parity. If the number of output bits is large enough, we can perform a hypothesis test according to Section 3. In this hypothesis test we need the probability distribution $P_0(\mathbf{Z_t})$. This distribution can be calculated by running through all different values of $\mathbf{U_t}, \mathbf{U_{t+\tau_1}}, \ldots, \mathbf{U_{t+\tau_{w-1}}}$. If d and w are large, the complexity for such a direct approach is too high. Then we can use slightly more advanced techniques based on building a trellis, that have much lower complexity.

The new basic distinguishing attack is summarized in Figure 2.

1. Find a weight w multiple of $g(x)$.
2. Calculate the distribution $P_0(\mathbf{Z}_t)$.
3. Calculate the length N we need to observe.
4. **for** $t = 0 \ldots N$
   $\mathbf{Z}_t = (z_t, z_{t+\tau_1}, \ldots, z_{t+\tau_{w-1}})$
   end for
5. Calculate $I = \sum_{t=0}^{N} \left(\log_2 \frac{P_0(\mathbf{Z}_t)}{1/2^w} \right)$.
6. **if** $(I > 0)$
   output "cipher" otherwise "random".

Fig. 2. Summary of the new basic distinguishing attack

4.3 Example of the Attack Applied on a Filter Generator

To really show the simplicity of the attack we will demonstrate with an example. We use an example from [16] in which we consider a three weight multiple from which the output is filtered by an 8-input, 2-resilient plateaued function.

$$f(x) = x_1 + x_4 + x_5 + x_6 + x_7 + x_1(x_2 + x_7) + x_2x_6 + x_3(x_6 + x_8) + \\ + x_1x_2(x_4 + x_6 + x_7 + x_8) + x_1x_3(x_2 + x_6) + x_1x_2x_3(x_4 + x_5 + x_8).$$

For a parity of weight three and using the notation from Section 4.2 we can write the vectors as

$$(z_t, z_{t+\tau_1}, z_{t+\tau_2}) = (f(\mathbf{U}_t), f(\mathbf{U}_{t+\tau_1}), f(\mathbf{U}_t + \mathbf{U}_{t+\tau_1})).$$

If we try all possible inputs to this function and determine the distribution $P_0(z_t, z_{t+\tau_1}, z_{t+\tau_2})$ we get Table 1. Since we know that the probability only

Table 1. The probability distribution $P_0(z_t, z_{t+\tau_1}, z_{t+\tau_2})$

$z_t, z_{t+\tau_1}, z_{t+\tau_2}$	$P_0(z_t, z_{t+\tau_1}, z_{t+\tau_2})$
000	$8320/2^{16}$
001	$8064/2^{16}$
010	$8064/2^{16}$
011	$8320/2^{16}$
100	$8064/2^{16}$
101	$8320/2^{16}$
110	$8320/2^{16}$
111	$8064/2^{16}$

depends on the parity of these vectors we translate these probabilities into binary probabilities. Using the binary probabilities we can calculate the bias, $\varepsilon = P(z_t + z_{t+\tau_1} + z_{t+\tau_2} = 0) = 4 \cdot \frac{8320}{2^{16}} - \frac{1}{2} = 7.8125 \cdot 10^{-3}$. As described before we use the thumb rule, $N \approx 1/\varepsilon^2$, for the number of output bits we need to observe in order to make a correct decision in the hypothesis test. This means that we need approximately 16384 bits to distinguish the cipher from a truly random source. Of course, we can use several weight three recursions (using squaring technique) and decrease the number of required bits. However, there are more powerful possibilities, as we will show in the next section.

4.4 Using More Than One Parity Check Equation

The distinguishing attack described in the previous section is in a very simple form. We can improve the performance by using a slightly more advanced technique. If we can find more than one low weight parity check equation, we can use them simultaneously to improve performance. Assume that we have the two parity check equations, $u_t + u_{t+\tau_1} + \ldots + u_{t+\tau_{w-1}} = 0$ and $u_t + u_{t+\tau_w} + \ldots + u_{t+\tau_{2w-2}} = 0$, giving rise to

$$\begin{array}{c} \mathbf{U}_t + \mathbf{U}_{t+\tau_1} + \ldots + \sum_{i=0}^{w-2} \mathbf{U}_{t+\tau_i} = \mathbf{0}, \\ \mathbf{U}_t + \mathbf{U}_{t+\tau_w} + \ldots + \sum_{i=w}^{2w-3} \mathbf{U}_{t+\tau_i} = \mathbf{0}. \end{array} \tag{7}$$

We introduce in this case

$$\mathbf{Z_t} = (z_t, z_{t+\tau_1}, \ldots, z_{t+\tau_{2w-2}}).$$

In this case two of the variables are totally determined by the other variables,

$$\mathbf{Z_t} = (f(\mathbf{U}_t), \ldots, f(\sum_{i=0}^{w-2} \mathbf{U}_{t+\tau_i}), f(\mathbf{U}_{t+\tau_w}), \ldots, f(\mathbf{U}_t + \sum_{i=w}^{2w-3} \mathbf{U}_{t+\tau_i})). \tag{8}$$

In a similar manner we can use more than two parity checks in the vectors. Assume that we have N parity check equations. Then we have N positions in the vector that are fully determined by other positions. This means a more skew distribution of the output vector in (8). For the particular case of two parity check equations, the algorithm is described in Figure 3.

1. Find two weight w multiples of $g(x)$.
2. Calculate the distribution $P_0(\mathbf{Z}_t)$.
3. Calculate the length, N we need to observe.
4. **for** $t = 0 \dots N$
   $\mathbf{Z}_t = (z_{t_t}, z_{t+\tau_1}, \dots, z_{t+\tau_{2w-2}})$
   end for
5. Calculate $I = \sum_{t=0}^{N} \left(\log_2 \frac{P_0(\mathbf{Z}_t)}{1/2^w} \right)$.
6. **if** $(I > 0)$
   output "cipher" otherwise "random".

Fig. 3. Summary of the new distinguishing attack using two parity check equations

4.5 Example of the Attack Applied on a Filter Generator, Cont'd

In this section we consider the same example as in Section 4.3, but we use two recursions as described in Section 4.4. Hence we observe the keystream vectors

$$(z_t, z_{t+\tau_1}, \dots, z_{t+\tau_4}) = (f(\mathbf{U}_t), f(\mathbf{U}_{t+\tau_1}), f(\mathbf{U}_t + \mathbf{U}_{t+\tau_1}), f(\mathbf{U}_{t+\tau_3}), f(\mathbf{U}_t + \mathbf{U}_{t+\tau_3})),$$

when $(\mathbf{U}_t, \mathbf{U}_{t+\tau_1}, \mathbf{U}_{t+\tau_3})$ runs through all values. In this case we use statistical distance, as defined in Section 3, in order to determine the number of vectors N we need to make a correct decision. The statistical distance is approximately $\varepsilon = 0.01172$ and hence we need $N \approx 1/\varepsilon^2 = 7282$ vectors to distinguish the key stream. We see that the result is a significant improvement. If we extend the reasoning and use three weight three recursions we get $\varepsilon = 0.01276$ and hence we need $N \approx 1/\varepsilon^2 = 6145$ vectors. The gain of using three recursions instead of two is smaller.

4.6 The Weight Three Attack

If we can find many multiples of weight three of a feedback polynomial we can simplify the description of our attack. With a d-input nonlinear function we write one parity check as

$$\mathbf{U}_t + \mathbf{U}_{t+\tau_1} + \mathbf{U}_{t+\tau_2} = \mathbf{0}.$$

If $\mathbf{U}_t = \mathbf{0}$ we notice that $\mathbf{U}_{t+\tau_1} = \mathbf{U}_{t+\tau_2}$. If this is the case, then obviously $z_{t+\tau_1} = z_{t+\tau_2}$ (we assume that $f(\mathbf{0}) = 0$). Now, we have m weight three parity checks, say

$$\begin{aligned} \mathbf{U}_t + \mathbf{U}_{t+\tau_1} + \mathbf{U}_{t+\tau_2} &= \mathbf{0}, \\ \mathbf{U}_t + \mathbf{U}_{t+\tau_3} + \mathbf{U}_{t+\tau_4} &= \mathbf{0}, \\ &\vdots \\ \mathbf{U}_t + \mathbf{U}_{t+\tau_{2m-1}} + \mathbf{U}_{t+\tau_{2m}} &= \mathbf{0}. \end{aligned}$$

Again assuming $\mathbf{U}_t = \mathbf{0}$ we see that we must have $z_{t+\tau_1} = z_{t+\tau_2},\ z_{t+\tau_3} = z_{t+\tau_4},\ \dots,\ z_{t+\tau_{2m-1}} = z_{t+\tau_{2m}}$. So, since $P(\mathbf{U}_t = \mathbf{0}) = 2^{-d}$ we will have

$$P(z_{t+\tau_1} = z_{t+\tau_2}, z_{t+\tau_3} = z_{t+\tau_4}, \dots, z_{t+\tau_{2m-1}} = z_{t+\tau_{2m}}) > 2^{-d}. \tag{9}$$

For a purely random sequence, however, this probability is 2^{-m}. It is important to note that when (9) holds for some t, it is very probable that $\mathbf{U}_t = \mathbf{0}$, i.e., *we have recovered a part of the key*. Since all output bits from the LFSR can be written as a linear combination of the initial state, $u_{t+t_i} = \sum_{i=0}^{r-1} a_i u_i,\ i = 1 \ldots d$ where $a_i \in \{0, 1\}$ are constants, we get d equations of the kind $u_{t+t_i} = \sum_{i=0}^{r-1} a_i u_i = 0$ for each $\mathbf{U}_t = \mathbf{0}$. Finding another value of t for which (9) holds gives more expressions describing the key. Since a full rank of the system of equations would only lead to the all zero solution, we need to guess at least one bit of the key. Then simple Gauss elimination can be applied to the system to deduce the other key bits. So we have described a *key recovery attack*. This attack has major consequences for any filter generator, as well as for nonlinear combining generators, and possibly also others. Basically, any filter generator of length r where the number of inputs d is smaller than $r/2$ can be broken very easily if we have access to a bit more than $2^{r/2}$ output symbols.

This leads to an attack as described in Figure 4.

1. Find m weight three multiples of $g(x)$.
2. Calculate the length N we need to observe.
3. **for** $t = 0 \ldots N$
   if $z_t = 0$ and
   $z_{t+\tau_1} = z_{t+\tau_2}$
   $z_{t+\tau_3} = z_{t+\tau_4}$
   $\vdots$
   $z_{t+\tau_{2m-1}} = z_{t+\tau_{2m}}$
   then assign $\mathbf{U}_t = \mathbf{0}$.
4. Guess at least one u_t and then recover $u_1, u_2, \ldots$ by linear algebra.

Fig. 4. Summary of the weight three key recovery attack

5 A Key Recovery Attack on LILI-128

In 2000 a project called NESSIE was initialized. The aim of this project was to collect a strong portfolio of cryptographic primitives. After a open call for proposals the submissions were thoroughly evaluated. One proposed candidate in the stream cipher category was called LILI-128 [3]. The cipher is very simple and its design is shift register based and uses a key of length 128 bits.

5.1 Description of LILI-128

LILI-128 has the structure of a filter generator. The only difference is that LILI-128 use an irregular clocking. LILI-128 consists of a first LFSR, called $LFSR_c$, that via a nonlinear function clocks a second LFSR, called $LFSR_d$, irregularly. The structured can be viewed in Figure 5. LILI-128 use a key length of 128 bits, the key is used directly to initialize the two binary LFSRs from left to right. Since

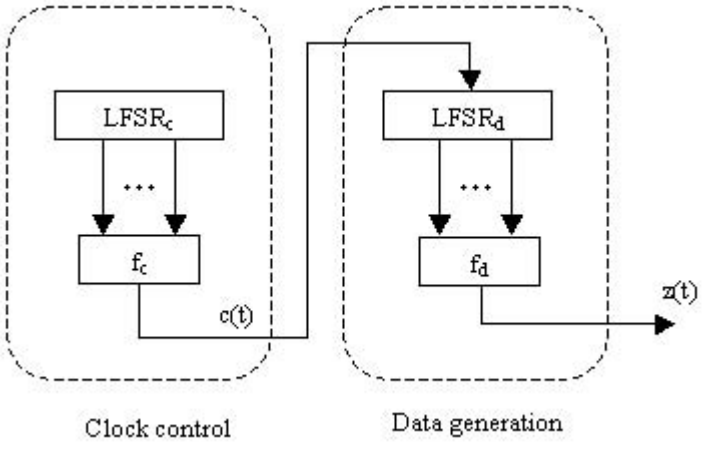

Fig. 5. Overview of LILI keystream generator

the the first shift register, $LFSR_c$ is a polynomial of length 39, the leftmost 39 bits of the key is used to initialize $LFSR_c$. The remaining 89 bits are used in the same manner to initialize $LFSR_d$. The feedback polynomial for $LFSR_c$ is given by

$$x^{39} + x^{35} + x^{33} + x^{31} + x^{17} + x^{15} + x^{14} + x^2 + 1 \ .$$

The Boolean function f_c takes two input bits from $LFSR_c$, namely the bit in stage 12 and the bit in stage 20 of the LFSR. The Boolean function f_c is chosen to be

$$f_c(x_{12}, x_{20}) = 2 \cdot x_{12} + x_{20} + 1 \ . \tag{10}$$

The output of this function is used to clock $LFSR_d$ irregularly. The reason for using irregular clocking [3], was that regularly clocked LFSRs are vulnerable to correlation and fast correlation attacks. The output sequence from f_c is denoted $c(t)$ and $c(t) \in \{1, 2, 3, 4\}$, i.e., $LFSR_d$ is clocked at least once and at most four times between consecutive outputs. On average, $LFSR_d$ is clocked $\bar{c} = 2.5$ times.

$LFSR_d$ is chosen to have a primitive polynomial of length 89 which produces a maximal-length sequence with a period of $P_d = 2^{89} - 1$. The feedback polynomial for $LFSR_d$ is

$$x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1 \ .$$

Ten bits are taken from $LFSR_d$ as input to the the function f_d, these bits are taken from the positions (0,1,2,7,12,20,30,44,65,80) of the LFSR. The function f_d is given as a truth table, see [3].

5.2 The Attack Applied on LILI-128

In this chapter we will give a description of how we turn our new ideas described in Section 4 into a key recovery attack on LILI-128. The different steps of our attack can be summarized as follows:

- First we find a multiple of low weight of the $LFSR_d$, see Section 4.1.
- Secondly, we guess the content of $LFSR_c$. For each guess we perform a distinguishing attack on the output keystream. If the guessed key is the correct, we will detect a certain a bias in the output.
- When we have found the correct starting state of $LFSR_c$, we recover the initial state of $LFSR_d$ by just applying some well known attack, e.g. a time-memory tradeoff attack, or the weight three attack described in this paper.

After calculation of one (or several) multiple(s) of the feedback polynomial of $LFSR_d$, our first step is to guess the initial state of $LFSR_c$. If we guess the correct key the clocking of $LFSR_d$ is correct and we should be able to detect some bias in the keystream. To detect this bias we apply our distinguishing attack on the keystream. If we instead made an incorrect guess, the output sequence will have properties like a random source. For each guess of $LFSR_c$ we need to make a decision whether this is the correct key or not. LILI-128 uses 10 bits as input to the Boolean function f_d. For a w-weight multiple we get

$$(z_t, z_{t+\tau_1}, \ldots, z_{t+\tau_{w-1}}) = (f_d(\mathbf{U}_t), f_d(\mathbf{U}_{t+\tau_1}), \ldots, f_d(\sum_{i=0}^{w-2} \mathbf{U}_{t+\tau_i})), \qquad (11)$$

where $\mathbf{U}_{t+\tau_l}$ is a column vector including the ten inputs to f_d. If we consider the fact that we have an irregularly clocked LFSR, not all of the terms in (4) will be used to produce an output bit. If so, we cannot use this relation. Thus we will need more keystream to be able to get the required number of valid vectors $\mathbf{Z_t}$ we want. As $LFSR_d$ is clocked on average 2.5 times between consecutive outputs we can expect that we need to increase the keystream by roughly a factor $(2/5)^{w-1}$. (This is valid for the case of one weight w parity check. If we would consider all weight w parity checks up to a certain length, we do not need to increase the keystream length at all in the case of irregular clocking.)

5.3 Results with Weight Three Multiple

We first use the method described in Section 4.1 on LILI-128 to find a multiple of weight three. In this case we have the degree of the original feedback $r = 89$ and $w = 3$, hence the degree of the multiple is approximately $2^{44.5}$. The complexity to find one multiple of weight three according to [9] is approximately 2^{50}. If we use the distinguishing attack described above on a regularly clocked $LFSR_d$, the bias is $\varepsilon = 1.953 \cdot 10^{-3}$ which is greater than we would usually expect. This means that we will need approximately $1/\varepsilon^2 = 2^{18}$ keystream bits to distinguish it from a stream of random data. We thus need about slightly more than $2^{44.5}/2.5$ bits of received sequence. The complexity for the attack is $2^{38} \cdot 2^{18} \cdot 2.5^2$ since we search through the initial states of $LFSR_c$, each of these states takes 2^{18} bits to distinguish. To get 2^{18} sample values, we need to use $2^{18} * 2.5^2 \approx 2^{21}$ parity checks in the case of irregular clocking. The total complexity is about 2^{60}.

5.4 Results with Weight Four Multiple

To find a multiple of weight four we use the same method as before. In this case we have the degree of the original feedback $r = 89$ and $k = 4$, hence the degree of the multiple is approximately $2^{29.67}$. The complexity to find this multiple is 2^{65}. The bias is calculated to $\varepsilon = 1.862 \cdot 10^{-3}$, see [20] for an explanations on why weight three and weight four multiples give almost the same bias. We will need approximately $1/\varepsilon^2 = 2^{18.14}$ keystream bits in the case when all parities are available. We use the same argument as in the 3-weight case for the uncertainty of positions actually appearing in the output stream and we get that we in total need 2^{22} bits. Since the degree of the polynomial is 2^{30} we will need about $2^{30}/2.5$ bits to detect the bias. The complexity for the attack is about 2^{61}.

5.5 Results with Weight Five Multiple

To find a multiple of weight five we need a degree of the multiple of approximately $2^{17.8}$ and it takes about 2^{50} time. With a multiple of weight five the bias decreases significantly to $\varepsilon = 7.182 \cdot 10^{-6}$ and hence we will need approximately $1/\varepsilon^2 = 2^{34}$ available checks. We need in total 2^{40} bits to perform the distinguishing attack. The complexity for the attack is around 2^{79}.

5.6 The Weight Three Attack

We can improve the results above by using several low weight parity checks as described in Section 4.4. We do not present the results here, but consider only the modified attack described in Section 4.6. In earlier sections we have stated that the multiples of weight three start to appear at degree $2^{44.5}$. If we use about 15 valid parity equations in the weight three attack, the probability that we make an incorrect decision is low. The total complexity for this attack is roughly 2^{53} since we still guess the contents of $LFSR_c$ 2^{38} times on average, and for each guess we need to test whether the set of equalities $z_{t+\tau_1} = z_{t+\tau_2}, \ldots$ is true. Since the probability of such an event occurring for the correct key is $> 2^{-10}$ we run through a bit more, say 2^{12} such t values. As all but one test correspond to the random case, the equalities will hold with probability $1/2$. Hence we need very few comparisons on average (say 2). We also need to include the fact that not all positions are present, a factor 2.5^2. The required keystream length to find 15 valid weight three parity checks is roughly 2^{47}.

5.7 Summary

In Table 2 we summarize our result. Note that the work to synchronize positions for each guessed $LFSR_c$ state was not considered in previous work, but can be done without increasing the overall complexity by choosing states in the order they appear in the $LFSR_c$ cycle, see also [19].

We compare with the best attacks so far, summarized in Table 3. Here we have recalculated the complexity of [22] to bit operations.

Table 2. The sequence length and the complexity for different weights, the basic attack is described in Section 5.2, and the weight three attack is described in Section 5.6

	Sequence length	Complexity
Weight three attack	2^{47}	2^{53}
Basic attack with weight 3	2^{43}	2^{60}
4	2^{29}	2^{61}
5	2^{34}	2^{79}

Table 3. Comparison of our attack with the best known attacks

Attack by	Our	[6]	[22]	[5]
Sequence length in bits	2^{47}	2^{18}	2^{46}	2^{60}
Attack complexity	2^{53}	2^{96}	2^{60}	"

6 Conclusions and Future Work

We have presented a new simple attack philosophy on filter generators and related ciphers. We demonstrated the efficiency by attacking LILI-128. We can recover the key using 2^{47} keystream bits with complexity around 2^{53}, an improvement compared to previous attacks. The weight three attack is a very powerful key recovery attack on any filter generator, if enough output symbols are available. It also applies to any filter generator with a weight three feedback polynomial, by the squaring method.

It is an open problem to examine whether these techniques can be applied on stronger designs like LILI-II and word-oriented stream ciphers.

Finally, we mention that related work has independently been done by Molland and Helleseth [20].

References

1. A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *Advances in Cryptology—EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. Springer-Verlag, 2000.
2. V. Chepyzhov, T. Johansson, and B. Smeets. A simple algorithm for fast correlation attacks on stream ciphers. In *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 181–195. Springer-Verlag, 2001.

3. A. Clark, E. Dawson, J. Fuller, J. Golic, H-J. Lee, William Millan, S-J. Moon, and L. Simpson. The LILI-128 keystream generator. In *Selected Areas in Cryptography—SAC 2000*, volume 2012 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.
4. D. Coppersmith, S. Halevi, and C.S. Jutla. Cryptanalysis of stream ciphers with linear masking. In M. Yung, editor, *Advances in Cryptology—CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 515–532. Springer-Verlag, 2002.
5. N. Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In D. Boneh, editor, *Advances in Cryptology—CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 176–194. Springer-Verlag, 2003.
6. N. Courtois and W. Meier. Algebraic attack on strem ciphers with linear feedback. In E. Biham, editor, *Advances in Cryptology—EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer-Verlag, 2003.
7. T. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley series in Telecommunication. Wiley, 1991.
8. P. Ekdahl and T. Johansson. Distinguishing attacks on SOBER-t16 and SOBER-t32. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 210–224. Springer-Verlag, 2002.
9. J.D. Golić. Computation of low-weight parity-check polynomials. *Electronic Letters*, 32(21):1981–1982, October 1996.
10. J.D. Golić. On the security of nonlinear filter generators. In D. Gollman, editor, *Fast Software Encryption'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 173–188. Springer-Verlag, 1996.
11. J.D. Golić, A. Clark, and E. Dawson. Inversion attack and branching. In J. Pieprzyk, R. Safavi-Naini, and J.Seberry, editors, *Information Security and Privacy: 4th Australasian Conference, ACISP'99*, volume 1587 of *Lecture Notes in Computer Science*, pages 88–102. Springer-Verlag, 1999.
12. J.D. Golić, A. Clark, and E. Dawson. Generalized inversion attack on nonlinear filter generators. 49(10):1100–1109, 2000.
13. T. Johansson and F. Jönsson. Fast correlation attacks based on turbo code techniques. In *Advances in Cryptology—CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 181–197. Springer-Verlag, 1999.
14. T. Johansson and F. Jönsson. Improved fast correlation attacks on stream ciphers via convolutional codes. In *Advances in Cryptology—EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 347–362. Springer-Verlag, 1999.
15. T. Johansson and F. Jönsson. A fast correlation attack on LILI-128. In *Information Processing Letters*, volume 81, pages 127–132, 2002.
16. S. Leveiller, G. Zémor, P. Guillot, and J. Boutros. A new cryptanalytic attack for pn-generators filtered by a boolean function. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography—SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 2003.
17. W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. In C.G. Günter, editor, *Advances in Cryptology—EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 301–316. Springer-Verlag, 1988.
18. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
19. H. Molland. Improved linear consistency attack on irregular clocked keystream generators. In *Fast Software Encryption 2004*.

20. H. Molland and T. Helleseth. An improved correlation attack against irregular clocked and filtered keystream generators. In *Advances in Cryptology—CRYPTO 2004*.
21. E. Pasalic. *On Boolean Functions in Symmetric-Key Ciphers*. PhD thesis, Lund University, Department of Information Technology, P.O. Box 118, SE–221 00, Lund, Sweden, 2003.
22. M-J.O. Saarinen. A time-memory tradeoff attack against LILI-128. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 231–236. Springer-Verlag, 2002.
23. T. Siegenthaler. Correlation-immunity of non-linear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, 30:776–780, 1984.
24. D. Wagner. A generalized birthday problem. In M. Yung, editor, *Advances in Cryptology—CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer-Verlag, 2002.

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