An Improved Correlation Attack Against Irregular Clocked and Filtered Keystream Generators

Håvard Molland and Tor Helleseth

The Selmer Center**
Institute for Informatics,
University of Bergen,
Norway

Abstract. In this paper we propose a new key recovery attack on irregular clocked keystream generators where the stream is filtered by a nonlinear Boolean function. We show that the attack is much more efficient than expected from previous analytic methods, and we believe it improves all previous attacks on the cipher model.

Keywords: Correlation attack, Stream cipher, Boolean functions, Irregular clocked shift registers.

1 Introduction

In this paper we present a new key recovery correlation attack on ciphers based on an irregular clocked linear feedback shift register (*LFSR*) filtered by a Boolean function. The cipher model we attack is composed of two components, the clock control generator and the data generator and is shown in Fig. 1.

- *The data generator sub system* consists of $\mathrm{LFSR_u}$ of length l_u and the nonlinear multivariate function f. The internal state of $\mathrm{LFSR_u}$ is filtered by a Boolean function f. The output from f is the high linear complexity bit stream $\mathbf{v}$.
- *The clock control sub system* consists of $\mathrm{LFSR_s}$ of length l_s where the output from $\mathrm{LFSR_s}$ is sent through the clock function $D()$. The output from $D()$ is the clock control sequence of integers, $\mathbf{c}$, which is used to clock $\mathrm{LFSR_u}$.

The effect of the irregular clocking is that $\mathbf{v}$ is irregularly decimated and the positions of the bits in the stream are altered. The result from this decimation is the keystream $\mathbf{z}$. The secret key in this cipher is the $(l_\mathrm{u}+l_\mathrm{s})$ initialization bits for $\mathrm{LFSR_u}$ and $\mathrm{LFSR_s}$ $(\mathbf{I}_\mathrm{u}, \mathbf{I}_\mathrm{s})$.

To attack this encryption scheme we need to know the positions the keystream bits $\mathbf{z}$ had in the stream $\mathbf{v}$ before $\mathbf{v}$ was irregularly decimated. The previous effective algorithms are not specially designed to attack irregular clocked and filtered

** This work was supported by the Norwegian Research Council.

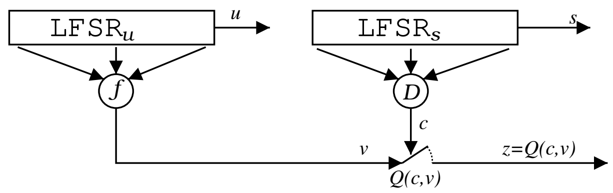

Fig. 1. The general cipher model we attack in this article

generators. But there exist effective attacks on the data generator sub system[6, 1, 10, 3, 4]. To deal with the irregular clocking, one of two techniques are often used:

1. **Do the attack on the data generator 2^{l_s} times**[7]. The attack is done one time for each guess for the 2^{l_s} possible initialization states for $\mathrm{LFSR_s}$. If the attack on the sub system has complexity $O(K)$ the full attack will have complexity $O(K \cdot 2^{l_s})$.
2. **Ignore the clock control generator**[3, 14, 4]. If the attack on the data generator subsystem needs M keystream bits, we can use the fact[14] that we know the original $\mathbf{v}$ position of every $2^{l_s} - 1$ bit in the keystream $\mathbf{z}$. Thus we can only use every $2^{l_s} - 1$ keystream bit in the attack, which means that we need $(2^{l_s} - 1) \cdot M$ keystream bits to succeed.

None of these techniques are optimal. The first one leads to large runtime complexity, the second leads to the need for a large number of keystream bits.

Our attack is not designed to attack the data generator subsystem only, but is especially aimed at irregular clocked and filtered keystream generators as one system. First we guess the initialization state $\mathbf{I}_s$ for $\mathrm{LFSR_s}$. From this we can reconstruct the positions the bits in $\mathbf{z}$ had in $\mathbf{v}$. Using the iteration algorithm from[11] this reconstruction is done using just a couple of operations per guess, exploiting the cyclic redundancies in $\mathrm{LFSR_s}$. This method is fully explained in Section 4.3. This method gives the guess $\mathbf{v}^* = (.., *, z_i, ..., z_j, ...*, ..., z_k, ..., *, ...)$, where z_i, z_j, z_k are some keystream bits and the stars are the deleted bits. Then we test $\mathbf{v}^*$ to see if it is likely that the stream is generated by the data generator subsystem $\mathrm{LFSR_u}$ and f. Hence, we only use a distinguisher test on the the $\mathbf{v}^*$ stream to decide if the guess for $\mathbf{I}_s$ is correct. This is easier than to actually decode the $\mathbf{v}^*$ stream to find $\mathbf{I}_u$**,** and then decide if we have found the correct $\mathbf{I}_s$. When $\mathbf{I}_s$ is determined, we can use one of the previous attacks on the data generator sub system to determine $\mathbf{I}_u$.

The distinguisher test is to evaluate a large number m of low weight parity check equations on the bit stream $\mathbf{v}^*$. All equations are derived from one multiple $h(x)$ of weight 4 of the generator polynomial $g_u(x)$. Surprisingly this test works much better than expected from previous evaluation methods. In previous correlation attacks, the Piling up lemma[9] is often used to calculate the correlation[1, 7, 6] which the algorithm must decode. Since our algorithm only uses a distinguisher on $\mathbf{v}^*$ we can use a correlation property of the function f

which gives much higher correlation between $\mathbf{v}^*$ and the keystream $\mathbf{z}$. Thus we need fewer parity check equations. This correlation property exists even if the function is correlation immune in the normal sense.

Our attack has complexity $O(2^{l_s} \cdot m)$, independently of the length of $\mathrm{LFSR_u}$. A cipher based on the model we attack in this paper is LILI-128. To attack the LILI-128 cipher our algorithm needs about 2^{23} parity check equations. In LILI-128, $l_s = 39$, thus the runtime for our attack is $2^{39+23} \approx 2^{62}$ parity checks, with virtually no precomputation. We have implemented and tested the attack, and it works on computers having under 300 MB of RAM, and needs only around 68 Mbyte of keystream data. The precomputation has low runtime complexity and is negligible. When $\mathbf{I}_s$ is found, we can use one of the previous algorithms to attack the data generator sub system.

A comparable previous correlation attack by Johansson and Jönsson is presented in [7]. The runtime for the attack is 2^{71} parity checks and the precomputations is 2^{79} table lookups. The keystream length is approximately 2^{30}. This attack uses the first technique to handle the irregular clocking.

Recently new algebraic attacks have been proposed by Courtois and Meier[3, 4]. This attack uses the second technique to handle the irregular clocking in LILI-128. Although the attack has an impressive runtime complexity $2^{31} \cdot C$ (an optimistic estimation for some unknown constant C), the attack needs about 2^{60} keystream bits to succeed, which is unpractical.

There is also a time-memory trade-off attack against LILI-128 by Markku-Juhani Olavi Saarinen[14]. This attack needs approximately $2^{51.4}$ bits of computer memory and 2^{46} keystream bits. The runtime complexity is claimed to be 2^{48} DES operations, which is not easy to compare with our runtime complexity. But the high use of computer memory and keystream bits also makes this attack unpractical.

2 A Correlation Property of Nonlinear Functions

Let $V = F_2^n$ and let f be a balanced Boolean function from V to F_2. We start by analyzing the boolean function $f(\mathbf{x})$ for a correlation property that we will use in the attack. A similar property is analyzed in [18] where they look at the nonhomomorphicity of functions. In this paper we identify the probability

$$p = P(f(\mathbf{x}_1) + f(\mathbf{x}_2) + f(\mathbf{x}_3) + f(\mathbf{x}_4) = 0 \mid \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 = \mathbf{0}) \quad (1)$$

which is crucial for our attacks success rate.

2.1 The Correlation Property

Let $q = 2^n$ and let $\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$ denote the inner product of $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$. Define the Walsh coefficients of f by

$$\hat{f}(\mathbf{a}) = \sum_{\mathbf{x} \in V} (-1)^{f(\mathbf{x}) + \mathbf{a} \cdot \mathbf{x}}.$$

Lemma 1. *Let f be a function from $V = F_2^n$ to F_2 and let $\mathbf{x}_i \in F_2^n$ for $i = 1, 2, 3, 4$. Let $q = 2^n$ and let N denote the number of solutions of*

$$\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 = \mathbf{0} \tag{2}$$
$$f(\mathbf{x}_1) + f(\mathbf{x}_2) + f(\mathbf{x}_3) + f(\mathbf{x}_4) = 0. \tag{3}$$

Then

$$N = \frac{q^3}{2} + \frac{1}{2q} \sum_{\mathbf{a} \in V} \hat{f}(\mathbf{a})^4. \tag{4}$$

Proof. Each term in the sum below gives a contribution $2q$ for each solution of the system of equations, and zero otherwise. Therefore, we have

$$\begin{aligned}
2qN &= \sum_{\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3,\mathbf{x}_4 \in V} \Big(\sum_{\mathbf{a} \in V} (-1)^{\mathbf{a}\cdot(\mathbf{x}_1+\mathbf{x}_2+\mathbf{x}_3+\mathbf{x}_4)}\Big)\Big(\sum_{y=0}^{1} (-1)^{y(f(\mathbf{x}_1)+f(\mathbf{x}_2)+f(\mathbf{x}_3)+f(\mathbf{x}_4))}\Big) \\
&= \sum_{\mathbf{a} \in V} \sum_{y=0}^{1} \sum_{\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3,\mathbf{x}_4 \in V} (-1)^{yf(\mathbf{x}_1)+\cdots+yf(\mathbf{x}_4)+\mathbf{a}\cdot\mathbf{x}_1+\cdots+\mathbf{a}\cdot\mathbf{x}_4} \\
&= \sum_{\mathbf{a} \in V} \sum_{y=0}^{1} \Big(\sum_{\mathbf{x} \in V} (-1)^{yf(\mathbf{x})+\mathbf{a}\cdot\mathbf{x}}\Big)^4 \\
&= q^4 + \sum_{\mathbf{a} \in V} \hat{f}(\mathbf{a})^4,
\end{aligned}$$

where the first term comes from the case $y = 0$ and $\mathbf{a} = \mathbf{0}$, and the last term from the case $y = 1$.

Corollary 1. *If $f(\mathbf{x})$ is a balanced function then the number of solutions N of the system of equations above is,*

$$N \geq \frac{q^3}{2} + \frac{q^3}{2(q-1)}.$$

Proof. Since $f(\mathbf{x})$ is balanced we obtain $\hat{f}(\mathbf{0}) = \sum_{\mathbf{x} \in V} (-1)^{f(\mathbf{x})} = 0$. It follows from Parseval's identity that the average value of $\hat{f}(\mathbf{a})^2$ is $\frac{q^2}{q-1}$. Hence, it follows from the Cauchy-Schwartz inequality that $\sum_{\mathbf{a} \in V} \hat{f}(\mathbf{a})^4 \geq (q-1)\frac{q^4}{(q-1)^2}$, which substituted in the lemma above gives the result.

Corollary 2. *The expected number of solutions N of the system of equations above is,*

$$E(N) = \frac{q^3}{2} + \frac{3q^2 - 2q}{2}.$$

Proof. An average estimate of N can be found as follows. When there exist two equal vectors $x_{i_1} = x_{i_2}$ in Equation (2), the two other vectors x_{i_3}, x_{i_4} will also be equal. When this occurs it follows that the Equation (3) will sum to zero.

This gives the unbalance that causes the high correlation. Equation (2) implies $\mathbf{x}_4 = \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3$ Then there are $q(q-1)(q-2)$ triples in $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ where all the $\mathbf{x}_i$'s are distinct and there are therefore $3q^2 - 2q$ triples with one or two pairs $\mathbf{x}_{i_1} = \mathbf{x}_{i_2}$. Using this fact and substituting Equation (2) into Equation (3), we can write

$$\begin{aligned} 2N &= \sum_{\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3 \in V} \sum_{y=0}^{1} (-1)^{y(f(\mathbf{x}_1)+f(\mathbf{x}_2)+f(\mathbf{x}_3)+f(\mathbf{x}_1+\mathbf{x}_2+\mathbf{x}_3))} \\ &= q^3 + \sum_{\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3 \in V} (-1)^{f(\mathbf{x}_1)+f(\mathbf{x}_2)+f(\mathbf{x}_3)+f(\mathbf{x}_1+\mathbf{x}_2+\mathbf{x}_3)} \\ &= q^3 + (3q^2 - 2q) + \sum_{\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3 \text{ distinct} \in V} (-1)^{f(\mathbf{x}_1)+f(\mathbf{x}_2)+f(\mathbf{x}_3)+f(\mathbf{x}_1+\mathbf{x}_2+\mathbf{x}_3)}. \end{aligned}$$

Since for an arbitrary function f we can expect that $f(\mathbf{x}_1)$, $f(\mathbf{x}_2)$, $f(\mathbf{x}_3)$, and $f(\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3)$ take on all binary quadruples approximately equally often when $\mathbf{x_1} \neq \mathbf{x_2} \neq \mathbf{x_3} \neq \mathbf{x_1}$, we expect in the average the last term to be 0. This implies the result.

Corollary 3. *Let f be an arbitrary balanced function, and let p denote the probability*

$$p = Prob(f(\mathbf{x}_1) + f(\mathbf{x}_2) + f(\mathbf{x}_3) + f(\mathbf{x}_4) = 0 \mid \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 = \mathbf{0}),$$

then p is expected to be $E(p) = \frac{1}{2} + \frac{3q-2}{2q^2}$ and its minimum is $p_{min} \geq \frac{1}{2} + \frac{1}{2(q-1)}$.

Proof. Since Equation (2) has q^3 solutions, it follows from Corollary 1 that the expected probability is equal to $E(p) = \frac{E(N)}{q^3} = \frac{1}{2} + \frac{3q-2}{2q^2}$. Further from Corollary 2 we obtain that the minimum is $p_{min} \geq (\frac{q^3}{2} + \frac{q^3}{2(q-1)})/q^3 = \frac{1}{2} + \frac{1}{2(q-1)}$.

Corollary 4. *Given a specific balanced function f, the probability*

$$p = Prob(f(\mathbf{x}_1) + f(\mathbf{x}_2) + f(\mathbf{x}_3) + f(\mathbf{x}_4) = 0 \mid \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 = \mathbf{0}),$$

is $p = \frac{1}{2} + \frac{\sum_{\mathbf{a} \in V} \hat{f}(\mathbf{a})^4}{2q^4}$

Proof. Using the N from Lemma 1 we get $p = \frac{N}{q^3} = \frac{1}{2} + \frac{\sum_{\mathbf{a} \in V} \hat{f}(\mathbf{a})^4}{2q^4}$

It is straightforward to extend Lemma 1 to compute the number of common solutions of the two equations

$$\begin{aligned} \mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_w &= \mathbf{0} \\ f(\mathbf{x}_1) + f(\mathbf{x}_2) + \cdots + f(\mathbf{x}_w) &= 0. \end{aligned}$$

and show that the corresponding probability

$$\text{Prob}(f(\mathbf{x}_1) + f(\mathbf{x}_2) + \cdots + f(\mathbf{x}_{w-1}) = 0 \mid \mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_{w-1} = \mathbf{0}),$$

equals $p = \frac{1}{2} + \frac{\sum_{\mathbf{a}\in V} \hat{f}(\mathbf{a})^w}{2q^w}$, which reduces to the result of Corollary 4 when $w = 4$.

In the case $w = 3$, we can calculate the expected value of a balanced Boolean function, with a given $f(\mathbf{0})$, to be $E(p) = \frac{1}{2} + \frac{3q-2}{2q^2}(-1)^{f(\mathbf{0})}$. This implies that the bias is the same for the case $w = 3$ as for $w = 4$. Similar arguments for equations with $w \geq 5$ show that these equations give too low correlation, which would lead to a high runtime complexity for our attack. It turns out that for $w = 3$ the attack needs much more keystream bits to succeed, see the Sections 4.1 and 5.2. Since the correlation bias is exactly the same for $w = 3$ and $w = 4$ it is optimal to use $w = 4$.

2.2 Analysis of Some Functions

In Table 1 we have analyzed some functions using Corollary 4. This correlation is surprisingly high. Let $p_{\mathrm{app}} = 0.53125$ be the best linear approximation to the LILI-128 function. Due to the design of the previous attacks[6, 7, 10] the channel noise has been independent of the stream $\mathbf{u}$ generated by LFSR$_{\mathrm{u}}$. Thus the Piling up lemma [9], $p_{\mathrm{pil}} = \frac{1}{2} + 2^{w-1}(\frac{1}{2} - p_{\mathrm{app}})^w$, is used to evaluate the crossover correlation $1 - p_{\mathrm{pil}}$ which the algorithms must be able to decode. Using the Piling up lemma for weight $w = 4$ equations, the correlation p_{pil} for LILI-128 will be $p_{\mathrm{pil}} = 0.50000763$. From Table 1 we have the correlation $p = 0.501862$.

Table 1. The probability $P(f(\mathbf{x}_1)+f(\mathbf{x}_2)+f(\mathbf{x}_3)+f(\mathbf{x}_4) = 0 \mid \mathbf{x}_1+\mathbf{x}_2+\mathbf{x}_3+\mathbf{x}_4 = \mathbf{0})$ calculated for some given functions. $E(p)$ is the expected correlation for given $q = 2^n$ and p is the actual correlation for the given function

Function	Number of inputs bits n	Best linear approximation.	$E(p)$	p
Geffe function	2	0.75	0.671875	0.625
LILI-128	10	0.53125	0.501464	0.501862
LILI-II	12	0.51367	0.500366	0.500190

The reason for the higher correlation, is that our attack only uses a distinguisher on the data generator sub system, and not a complete decoder. Hence, in our key recovery attack on the clock control system, we can use Corollary 4 from Section 2.1 to calculate the correlation. To test the corollary we generated 2000 random and balanced Boolean tables for $n = 10$, and calculated the average correlation. The result was that the average p was 0.501466 which is close to the theoretical expected $E(p) = 0.5001464$.

3 A General Model

Here we define a general model for irregular clocked and filtered stream ciphers, and some well known properties for the model.

3.1 General Model

Let $g_{\mathrm{u}}(x)$ and $g_{\mathrm{s}}(x)$ be the feedback polynomials for the shift registers $\mathrm{LFSR_u}$ of length l_{u} and $\mathrm{LFSR_s}$ of length l_{s}. We let $\mathbf{I}_{\mathrm{s}} = (s_0, s_1, ..., s_{l_{\mathrm{s}}-1})$ and $\mathbf{I}_{\mathrm{u}} = (u_0, u_1, ..., u_{l_{\mathrm{u}}-1})$ be the initialization states for $\mathrm{LFSR_s}$ and $\mathrm{LFSR_u}$. The initialization states $(\mathbf{I}_{\mathrm{s}}, \mathbf{I}_{\mathrm{u}})$ define *the secret key* for the given cipher system.

From $g_{\mathrm{s}}(x)$ we can calculate a clock control sequence $\mathbf{c}$ in the following way. Let $c_t = D(L_{\mathrm{s}}^t(\mathbf{I}_{\mathrm{s}})) \in \{a_1, a_2, ..., a_A\}$, $a_j \geq 0$, be a function where the input $L_{\mathrm{s}}^t(\mathbf{I}_{\mathrm{s}})$ is the inner state of $\mathrm{LFSR_s}$ after t feedback shifts and A is the number of values that c_t can take. Let p_j be the probability $p_j = \mathrm{Prob}(c_t = a_j)$.

$\mathrm{LFSR_u}$ produces the stream $\mathbf{u} = (u_0, u_1, ...)$ which is filtered by f. The output from f is $v_k = f(u_{k+i_0}, u_{k+i_1}, ..., u_{k+i_{n-1}})$, or the equivalent $v_k = f(L_{\mathrm{u}}^k(\mathbf{I}_{\mathrm{u}}))$. The clock c_t decides how many times $\mathrm{LFSR_u}$ is clocked before the output bit v_k is taken as keystream bit z_t. Thus the keystream z_t is produced by $z_t = v_{k(t)}$, where $k(t)$ is the total sum of the clock at time t, that is $k(t) \leftarrow k(t-1) + c_t$. This gives the following definition for the clocking of $\mathrm{LFSR_u}$.

Definition 1. *Given bit stream* $\mathbf{v}$ *and clock control sequence* $\mathbf{c}$*, let* $\mathbf{z} = Q(\mathbf{c}, \mathbf{v})$ *be the function that generates* $\mathbf{z}$ *of length* M *by*

$$Q(\mathbf{c}, \mathbf{v}) : \, z_t \leftarrow v_{k(t)}, \, 0 \leq t < M$$

where $k(t) = \sum_{j=0}^{t} c_j - 1$.

If $a_j \geq 1, \;\; 1 \leq j \leq A$, the function $Q(\mathbf{c}, \mathbf{v})$ can be considered as a deletion channel with input $\mathbf{v}$ and output $\mathbf{z}$. The deletion rate is

$$P_{\mathrm{d}} = 1 - \frac{1}{\sum_{j=1}^{A} p_j a_j}. \tag{5}$$

The $D()$ function described above can in this model be among others the shrinking generator, the step-1/step-2 generator and the stop and go generator. Next we define the (not complete) reverse of Definition 1.

Definition 2. *Given the clock control sequence* $\mathbf{c}$ *and keystream* $\mathbf{z}$*, let the function* $\mathbf{v}^* = Q^*(\mathbf{c}, \mathbf{z})$ *be the (not complete) reverse of* Q*, defined as*

$$Q^*(\mathbf{c}, \mathbf{z}) : \, v^*_{k(t)} \leftarrow z_t, \, 0 \leq t < M,$$

where $k(t) = \sum_{j=0}^{t} c_j - 1$*, and* $v_k = *$ *for the entries* k *in* $\mathbf{v}^*$ *where* v_k^* *is deleted. When this occurs we say that* v_k^* *is not defined.*

The length of $\mathbf{v}^*$ will be $N^* = \sum_{j=0}^{M-1} c_j$. Given a stream $\mathbf{z}$ of length M, the expected length N of the stream $\mathbf{v}$ is

$$\mathrm{E}(N) = \frac{M}{(1 - P_{\mathrm{d}})} = M \sum_{j=1}^{A} p_j a_j. \tag{6}$$

Note that the only difference between this definition and Definition 1, is that $\mathbf{v}$ and $\mathbf{z}$ have switched sides. Thus $Q^*(\mathbf{c}, \mathbf{z})$ is a reverse of $Q(\mathbf{c}, \mathbf{v})$. But since some bits are deleted, the reverse is not complete and we get the stream $\mathbf{v}^*$.

The probability for a bit v_k^* being defined is $\mathrm{Prob}(v_k^*) = 1 - P_\mathrm{d}$. This happens when $k = k(t)$ holds for some t, $0 \le t < M$. It follows that the sum $v_k^* + v_{k+j_1}^* + ... + v_{k+j_{w-1}}^*$ will be defined if and only if all of the bits in the sum are defined. Thus the sum will be defined for given k in $\mathbf{v}^*$ with probability

$$P_\mathrm{def} = (1 - P_\mathrm{d})^w. \tag{7}$$

4 The Attack

4.1 Equations of Weight 4

To succeed with our attack we need to find exactly one weight 4 equation

$$\lambda_\mathrm{u} : \; u_k + u_{k+j_1} + u_{k+j_2} + u_{k+j_3} = 0 \tag{8}$$

that holds over all $\mathbf{u}$ generated by $\mathrm{LFSR_u}$ for $k \ge 0$. This corresponds to finding a multiple $h(x) = a(x)g_\mathrm{u}(x)$ of weight 4. There exist several algorithms for finding such a multiple, see among others [13, 2, 5, 17, 12].

In this paper we use the fast search algorithm in [12, 11], which is a modified version of the David Wagner's Generalized Birthday Algorithm[17]. If the stream $\mathbf{u}$ has length N, this algorithm has runtime complexity $O(N \log N)$ and memory complexity $O(N)$, where N is of order $2^{l_\mathrm{u}/3}$. The algorithm is effective in practice, and we have succeeded in finding multiples of the generator polynomial of high degree, see Section 6.3 for an example. We refer to Appendix C in [11] for the details for this search algorithm.

Next, we let the input vector $\mathbf{x}_k$ to the Boolean function $f(\mathbf{x})$ be

$$\mathbf{x}_k = (u_{k+i_0}, u_{k+i_1}, ..., u_{k+i_{n-1}}), \tag{9}$$

where $(i_0, i_1, ..., i_{n-1})$ defines the tapping positions from the internal state $L_\mathrm{u}^k(\mathbf{I}_\mathrm{u})$ of $\mathrm{LFSR_u}$ after k feedback shifts. Substituting the vector (9) into the Equation (8) we have that $\mathbf{x}_k + \mathbf{x}_{k+j_1} + \mathbf{x}_{k+j_2} + \mathbf{x}_{k+j_3} = \mathbf{0}$ always holds for $k \ge 0$. Since $v_k = f(\mathbf{x}_k)$ we have from Corollary 4 that the equation

$$\lambda_\mathrm{v} : \; v_k + v_{k+j_1} + v_{k+j_2} + v_{k+j_3} \approx 0, \tag{10}$$

will hold for $k \ge 0$ with probability $p = \frac{1}{2} + \frac{\sum_{\mathbf{a} \in V} \hat{f}(\mathbf{a})^4}{2q^4}$.

Remark 1. In [8] the multiple of $g_\mathrm{u}(x)$ of weight $w = 3$ is exploited to define an iterative decoding attack on regularly clocked LFSRs filtered by Boolean functions. The constrained system

$$\sum_{a=0}^{w-1} \mathbf{x}_{k+j_a} = \mathbf{0} \tag{11}$$
$$z_{k+j_a} = f(\mathbf{x}_{k+j_a}),\; 0 \le a < w$$

is analyzed. This system is similar to the one we use in this paper, but it is used differently. Since there are limited solutions to this system, the *a posteriori* probabilities for each of the *input bits* $(u_{k+j_a+i_0}, u_{k+j_a+i_1}, ..., u_{k+j_a+i_{n-1}})$ in $\mathbf{x}_{k+j_a}$ can be calculated. Then these probabilities are put into a Gallager like probabilistic decoding algorithm(SOJA) which outputs $\mathbf{I}_{\mathrm{u}}$. However the correlation property in Corollary 4 is neither identified or exploited in [8].

4.2 Naive Algorithm

Let $\hat{\mathbf{I}}_{\mathrm{s}}$ be a guess for the initialization state $\mathbf{I}_{\mathrm{s}}$. Given the keystream $\mathbf{z}$ of length M, we generate $\hat{c}_t = D(L_{\mathrm{s}}^t(\hat{\mathbf{I}}_{\mathrm{s}}))$, $0 \le t \le M$ and $\hat{\mathbf{v}}^* = Q^*(\hat{\mathbf{c}}, \mathbf{z})$ of length $N \approx \sum_{t=0}^{M-1} \hat{c}_t^i$. Then we test if $\hat{\mathbf{v}}^*$ is likely to have been generated by $\mathrm{LFSR}_{\mathrm{u}}$ using the following method.

Find m entries in $\hat{\mathbf{v}}^*$ where the equation is defined. From this we get a set of m equations. We test the m equations, and let the metric for the guess be the number of equations that hold. When we have the correct guess for $\mathbf{I}_{\mathrm{s}}$ we expect pm of the equations to hold, where p is calculated using Corollary 4. Thus, this is a maximum likelihood decoding algorithm.

The runtime complexity for the attack will be of order $2^{l_{\mathrm{s}}} \cdot (m + N)$, since we have to generate the bit stream $\hat{\mathbf{v}}^*$ of length N for each of the $2^{l_{\mathrm{s}}}$ guesses. In a real attack, N will be a large number and the naive algorithm will have very high runtime complexity.

4.3 Some Observations

If we use the technique in the previous section the attack has the runtime $2^{l_{\mathrm{s}}} \cdot (m + N)$. In [11, Sec. 3.3] two important observations were made that reduce the complexity down to $2^{l_{\mathrm{s}}} \cdot m$. Since $N \gg m$, these observations will speed up the attack considerably. We start with an initial guess $\mathbf{I}_{\mathrm{s}}^0 = (1, 0, ..., 0)$ and let the i'th guess be the internal state of $\mathrm{LFSR}_{\mathrm{s}}$ after i feedback shifts, that is $\mathbf{I}_{\mathrm{s}}^i = L_{\mathrm{s}}^i(\mathbf{I}_{\mathrm{s}}^0)$.

Let $\mathbf{c}^i = (c_0^i, c_1^i, ..., c_{M-1}^i)$ be the i'th guess for the clock control sequence defined by $c_t^i = D(L_{\mathrm{s}}^{i+t}(1, 0, ..., 0))$, $0 \le t < M$. Let $\mathbf{v}^i = Q^*(\mathbf{c}^i, \mathbf{z})$ be the corresponding guess for $\mathbf{v}^*$ of length $N_i = \sum_{t=0}^{M-1} c_t^i$. We can now give a iterative method for generating $\mathbf{v}^{i+1}$ from $\mathbf{v}^i$.

Lemma 2. *We can transform* $\mathbf{v}^i$ *into* $\mathbf{v}^{i+1} = Q^*(\mathbf{c}^{i+1}, \mathbf{z})$ *using the following method: Delete the first* c_0^i *entries* $(*, ..., *, z_0)$ *in* $\mathbf{v}^i$*, append the* $c_{M-1}^{i+1} = c_M^i$ *entries* $(*, ..., *, z_M)$ *at the end, and replace* z_t *with* z_{t-1} *for* $1 \le t \le M$*.*

Proof. See Appendix B.1 in [11].

Lemma 2 shows that we can generate each $\mathbf{v}^i$ using just a few operations instead of N operations, when implemented properly (See Appendix A.1 for the implementation details). This gives a fast method for generating all possible guesses for $\mathbf{v}^*$ given a keystream $\mathbf{z}$. But using this lemma we still have to search for m

entries in $\mathbf{v}^*$ where the equations are defined. Since on average we must search through $1/P_{\text{def}}$ entries in $\mathbf{v}^*$ per equation, we want to avoid this search. In the next theorem we show how this can be done. The theorem proves that we can reuse the equation set for $\mathbf{v}^i$ in $\mathbf{v}^{i+1}$.

Theorem 1. *If the sum*

$$v_k + v_{k+k_1} + ... + v_{k+k_{w-1}} = z_t + z_{t+j_1} + ... + z_{t+j_{w-1}} = \gamma_{\mathbf{z},t}$$

is defined over $\mathbf{v}^i$*, then the sum*

$$v_{k-c_0^i} + ... + v_{k+k_{w-1}-c_0^i} = z_{t-1} + z_{t+j_1-1} ... + z_{t+j_{w-1}-1} = \gamma_{\mathbf{z},t-1}$$

is defined over $\mathbf{v}^{i+1}$.

Proof. See Appendix B.2 in [11].

The main result from this theorem is that the equation set defined over $\mathbf{v}^i$ will be defined over $\mathbf{v}^{i+1}$ when we shift the equations c_0^i entries to the left over $\mathbf{v}^{i+1}$. This means that we can just shift the equations one entry to the left over $\mathbf{z}$, and we will have a sum that is defined for the guess $\hat{\mathbf{I}}_{\text{s}} = D(L_{\text{s}}^{i+1}(1,0,...,0)$. Thus, the theorem shows that we can avoid a lot of computations if we let the i'th guess for the inner state of LFSR$_{\text{s}}$ be $L_{\text{s}}^i(1,0,...,0)$.

Remark 2. To use the lemma and theorem above we do not put the *actual bit values* z_t and restore them to the position $k(t)$ in $\mathbf{v}^*$ given by $Q^*(\mathbf{c},\mathbf{z})$. Instead we store the *index* z_t (the pointer to the position t in $\mathbf{z}$) in $v_{k(t)}$. This means that $v^*_{k(t)}$ holds the position t, which the keystream bit z_t have in $\mathbf{z}$. But when we evaluate an equation we use the indices to put in the actual bit values.

4.4 An Efficient Algorithm

Assume we have found an equation $\lambda_{\text{v}} : v_k + v_{k+j_1} + v_{k+j_2} + v_{k+j_3} \approx 0$. The equation holds over $\mathbf{v}$ with probability p calculated using Corollary 4. Let the first guess for the initialization state for $\mathbf{s}$ be $\mathbf{I}_{\text{s}}^0 = (1,0,0,...,0)$, generate $\mathbf{c}^0$ by $c_t^0 = D(L_{\text{s}}^t(1,0,...0))$, $t < M$, and $\mathbf{v}^0 = Q^*(\mathbf{c}^0,\mathbf{z})$. Next we try to find m entries $(k_1,k_2,...,k_m)$ in $\mathbf{v}^0$ where the equation λ_{v} is defined. From this we get the equation set

$$\begin{array}{rl}
v^0_{k_1} + v^0_{k_1+j_1} + v^0_{k_1+j_2} + v^0_{k_1+j_3} & \approx 0 \\
v^0_{k_2} + v^0_{k_2+j_1} + v^0_{k_2+j_2} + v^0_{k_2+j_3} & \approx 0 \\
\vdots & \vdots \\
v^0_{k_m} + v^0_{k_m+j_1} + v^0_{k_m+j_2} + v^0_{k_m+j_3} & \approx 0.
\end{array} \tag{12}$$

Since every $v_{k_x+j_y}$ in this equation set is defined in $\mathbf{v}^0$ and $z_t = v_{k(t)}$, we can replace $v_{k_x+j_y}$ with the corresponding bit z_{t_x} from the keystream $\mathbf{z}$. Thus, $\mathbf{v}^0$ is a sequence of pointers to $\mathbf{z}$ and we can write the equations over $\mathbf{z}$ as the equation set Ω :

$$\begin{aligned}
z_{t_{1,1}} + z_{t_{1,2}} + z_{t_{1,3}} + z_{t_{1,4}} &\approx 0 \\
z_{t_{2,1}} + z_{t_{2,2}} + z_{t_{2,3}} + z_{t_{2,w}} &\approx 0 \\
\vdots \qquad\qquad & \quad \vdots \\
z_{t_{m,1}} + z_{t_{m,2}} + z_{t_{m,3}} + z_{t_{m,w}} &\approx 0.
\end{aligned} \tag{13}$$

We are now finished with the precomputation. Let $metric_{\text{best}}$ be the number of equations in Ω that hold. We iterate as follows:

Input The keystream $\mathbf{z}$ of length M, the equation λ, the equation set Ω, the index sequence $\mathbf{v}^0$, the states $L^0(1,0,...0)$ and $L^M(1,0...,0)$, and let $i \leftarrow 0$.

1. Calculate $c_{M-1}^{i+1} = c_M^i = D(L_{\text{s}}^{M+i}(1,0,...,0))$.
2. Use Lemma 2 to generate $\mathbf{v}^{i+1} = Q^*(\mathbf{c}^{i+1}, \mathbf{z})$ and lower all indexes in the equation set Ω by one. Theorem 1 guarantees that the equations are defined over $\mathbf{v}^{i+1}$.
3. If the first equation in Ω gets a negative index, then remove the equation from Ω. Find a new index at the end of $\mathbf{v}^{i+1}$ where λ is defined, and add the new equation over $\mathbf{z}$ to Ω.
4. Calculate *metric* as the number of equations in Ω that hold.
5. If $metric_{\text{best}} > metric$, set $metric_{\text{best}} \leftarrow metric$ and $\mathbf{I}_{\text{s}}^i = L_{\text{s}}^i(10,0,...0)$.
6. Set $i \leftarrow i+1$ and go to step 1.
7. Output $\mathbf{I}_{\text{s}}^i$ as the initialization state for LFSR_{s}.

Remark 3. The algorithm is presented this way to make it readable and to show the basic idea. To reach the complexity $O(2^{l_{\text{s}}} \cdot m)$ a few technical details on the implementation of the algorithm are needed. These details are given in Appendix A.

5 Theoretical Properties

5.1 Success Formula

We can let an (unusual) encoder be defined by removing the Boolean function from the cipher. Then we can use coding theory to evaluate the attack. Let the initialization state $\mathbf{I}_{\text{s}}$ for LFSR_{s} define the information bits in such a system.

Let $\mathbf{y} = (y_0, y_1, ..., y_{M-1})$ be the (not filtered) irregular clocked stream from LFSR_{u}, that is $\mathbf{y} = Q(\mathbf{c}, \mathbf{u})$ and $c_t = D(L_{\text{s}}^t(\mathbf{I}_{\text{s}}))$. Then the bitstream $\mathbf{y}$ defines the codeword that is sent over a noisy channel. Let the keystream $\mathbf{z} = Q(\mathbf{c}, \mathbf{v})$ (the filtered version of $\mathbf{y}$) be the received codeword.

Assume we have the wrong guess for $\mathbf{I}_{\text{s}}$, then approximately $m/2$ of the equations in the set (13) will hold. Now assume we have have guessed the correct $\mathbf{I}_{\text{s}}$. According to the observation in Section 2.1 the equations in the set (13) will hold with probability $p = \frac{1}{2} + \sum_{\mathbf{a} \in V} \hat{f}(\mathbf{a})^4 / 2q^4$, independently of the initialization bits $\mathbf{I}_{\text{u}}$.

Let p define the channel 'noise'. The uncertainty is defined by $H(p) = -p\log p - (1-p)\log(1-p)$, and the channel capacity is given by $C(p) = 1 - H(p)$. We can approximate $C(p)$ with $C(p) \approx 2(p - \frac{1}{2})^2 / \ln 2$. Following Shannon's noisy coding theorem we can set up this bound for success.

Proposition 1. *The attack will succeed with probability* $> \frac{1}{2}$ *if the number of parity check equations* m *is*

$$m > m_0 = \frac{l_\mathrm{s}}{C(p)} \approx \frac{0.347 l_\mathrm{s}}{(p - \frac{1}{2})^2}$$

where $p \approx \frac{1}{2} + \sum_{y \in V} \hat{f}(y)^4 / 2q^4$ *and* $q = 2^n$*, where* n *is the number of input bits in* $f(\mathbf{x})$*.*

When m is close to $2 \cdot m_0$ we expect the probability for success to be close to 1, see [15]. The simulations of our algorithm show that if we set $m = 2.1 \cdot m_0$ the success rate is approximately 99%.

5.2 Keystream Length

If the generator polynomial $g_\mathrm{u}(x)$ has weight $w > 4$, we must find a multiple $h(x)$ of $g_\mathrm{u}(x)$ of weight 4 and a degree l_h. We need at least the $\mathbf{v}$ stream to be of length l_h. In addition, to find m entries in $\mathbf{v}$ where the equation is defined $\mathbf{v}$ must at least have length

$$N > l_\mathrm{h} + m/P_\mathrm{def}. \tag{14}$$

From the expectation (6) of N we get $E(M) = N(1 - P_\mathrm{d}) = (1 - P_\mathrm{d})l_\mathrm{h} + m/(1 - P_\mathrm{d})^3$, which proves the following proposition:

Proposition 2. *Let an equation over* $\mathbf{v}$ *be defined by* $h(x)$ *of weight* 4 *and degree* l_h*. To obtain an equation set* Ω *of* m *equations over* $\mathbf{z}$*, the length of the* $\mathbf{z}$ *stream must be*

$$M > (1 - P_\mathrm{d})l_\mathrm{h} + m/(1 - P_\mathrm{d})^3. \tag{15}$$

The keystream length M depends on the number of equations m, the deletion rate P_d and the degree l_h of $h(x)$. The degree l_h is then again highly dependent on the search algorithm we use to find $h(x)$. When we use the search algorithm in [11, 17] the degree l_h of $g_\mathrm{h}(x)$ will be of order $l_\mathrm{h} = 2^{(2+l_\mathrm{u})/3}$, which is close to the theoretical expected degree $2^{l_\mathrm{u}/(w-1)}$ [5] for $w = 4$.

5.3 Runtime Complexity

The runtime complexity for our attack is

$$O(2^{l_\mathrm{s}} \cdot m) = O(\frac{2^{l_\mathrm{s}} \cdot l_\mathrm{s}}{(p - \frac{1}{2})^2}) \tag{16}$$

parity check tests, where p is calculated using Corollary 4. Note that the runtime is independent of the length l_u of $\mathrm{LFSR_u}$.

5.4 Memory Complexity

If we implement the attack directly as described in Sections 4.3 and 4.4 the algorithm will need around $32N + 4 * 32m$ bits of computer memory. The reason for the $32N$ term is that $\mathbf{v}^i = z_0, *, *, z_1, z_2, ..., *, z_{M-1}$ of length N is a sequence of pointers of 32 bits. In appendix A.2 we show how we can store $\mathbf{v}^i$ using N memory bits without affecting the runtime complexity. The total amount of memory *bytes* needed is then

$$\frac{N}{8} + 16m \tag{17}$$

6 Simulations of the Attack

The LILI-128 cipher[16] is based on the general model we attack in this paper. To be able to compare our attack with previous attacks, we have tested the attack on this cipher.

6.1 The LILI-128 cipher

In the LILI cipher the clock control generator is defined by

$$g_{\mathrm{s}}(x) = x^{39} + x^{35} + x^{33} + x^{31} + x^{17} + x^{15} + x^{14} + x^2 + 1,$$

and $c_t = D(s_{t+12}, s_{t+20}) = 1 + s_{t+12} + 2s_{t+20}$. The data generator sub system is

$$g_{\mathrm{u}}(x) = x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1,$$

and $v_k = f(u_k, u_{k+1}, u_{k+3}, u_{k+7}, u_{k+12}, u_{k+20}, u_{k+30}, u_{k+44}, u_{k+65}, u_{k+80})$, defined by a Boolean table of size 1024. Further on we get $P_{\mathrm{d}} = 0.6$, and $P_{\mathrm{def}} = 0.0256$ for $w = 4$, and $p = 0.501862$. The number of keybits in the secret key $(\mathbf{I}_{\mathrm{s}}, \mathbf{I}_{\mathrm{u}})$ is $39 + 89 = 128$.

6.2 Simulations

We have done the simulations on some versions of the LILI-128 cipher with LFSRs of different lengths to empirically verify the success formula in Section 5.1. See Table 2 for the simulations. Note that we use the full size $\mathrm{LFSR}_{\mathrm{u}}$ from the LILI cipher in the three attacks in the bottom of the table. For $l_s = 11$ and $p = 0.501862$ we get $m_0 = 1.1 \cdot 10^6$.

We have implemented the attack in C code using the Intel icc compiler on a Pentium IV processor. Using the full 32-bit capability and all the implementation tricks explained in Appendix A our implementation uses only approximately 7 cycles per parity check test. Hence the algorithm works fast in practice and will take $7 \cdot 2^{l_s} m$ processor cycles.

Each attack is run 100 times, and the table shows that the estimated success rate holds and that the algorithm is efficient.

Table 2. We have tested the attack on the LILI-128 Boolean function with $p = 0.501862$. Note that the runtime for finding $\mathbf{I}_\mathrm{s}$ is independent of the length l_u of LFSR_u, and the length M of the keystream. The attack on a full LFSR_u of length 89 and reduced LFSR_s of length 11 took 12 seconds

l_s	l_u	Keystream length M	Successes out of 100	m	Runtime	$2^{l_s} \cdot m$
11	60	$2^{24,1}$	59	m_0	6 sec.	2^{31}
11	60	$2^{25,1}$	100	$2.2 \cdot m_0$	13 sec.	2^{32}
11	40	$2^{24,0}$	51	m_0	6 sec.	2^{31}
11	40	$2^{25,0}$	100	$2.2 \cdot m_0$	13 sec.	2^{32}
10	**89**	$\mathbf{2^{29}}$	99	$2.1 \cdot m_0$	6 sec	2^{32}
11	**89**	$\mathbf{2^{29}}$	99	$2.1 \cdot m_0$	12 sec	2^{33}
12	**89**	$\mathbf{2^{29}}$	99	$2.1 \cdot m_0$	24 sec	2^{34}

6.3 A Complete Attack on LILI-128

Preprocessing For the LILI cipher, we have found a multiple $h(x) = a(x)g_\mathrm{u}(x)$ which corresponds to the recursion $u_t + u_{t+139501803} + u_{t+210123252} + u_{t+1243366916} = 0$ and we have that

$$\mathrm{Prob}(v_t + v_{t+139501803} + v_{t+210123252} + v_{t+1243366916} = 0) = 0.501862. \qquad (18)$$

This precomputation took only 5 hours and 40 Gbyte hard disk space. We see that $l_\mathrm{h} = 1243366916$.

Finding $\mathbf{I_s}$ We have $p = 0.501862$, and $m_0 = 39/C(0.501862) \approx 3.9 \cdot 10^6 \approx 2^{21.9}$. To be almost sure to succeed we use $m = 2.1 m_0$ equations. Hence, the runtime for attacking LILI-128 is

$$2^{39} \cdot 2^{23} = 2^{62}$$

parity checks. Using our implementation this corresponds to $2^{62} \cdot 7$ processor cycles. Using Proposition 2 with $P_\mathrm{d} = 0.6$ we need a keystream of length $M \approx 2^{29}$. The attack needs about 290 Mbyte of RAM. It can easily be parallelized and distributed among processors with virtually no overhead, since there is no need for communcation between the processor, and no need for shared memory. If we have 1024 Pentium IV 2.53 GHz processors, each having access to about 290 MB of memory, the attack would take about 4.5 months using 68 Mbyte of keystream data.

Finding $\mathbf{I_u}$ when $\mathbf{I_s}$ is known Our attack only finds the initialization bits $\mathbf{I}_\mathrm{s}$ for LFSR_s. It is possible to combine the Quick Metric from [12] with the previous attack against LILI in [7] to find $\mathbf{I}_\mathrm{u}$ when $\mathbf{I}_\mathrm{s}$ is given. Since this is not the scope of this paper we will not go into details, and we refer to [7, 12] for the exact description. The preprosessing stage will have complexity of order $2^{44.7}$ memory lookups, and runtime complexity of order $2^{42.5}$ parity checks. The complexity for the method above is much lower than the complexity for finding $\mathbf{I}_\mathrm{s}$ and will therefore have little effect on the overall runtime for a full attack.

7 Conclusion

We have proposed a new key recovery correlation attack on irregular clocked keystream generators where the stream is filtered by a nonlinear Boolean function. Our attack uses a correlation property of Boolean functions, that gives higher correlation than previous methods. Thus we need fewer equations to succeed. The property holds even if the function is correlation immune. Using this property together with the iteration techniques from [11] we get a low runtime and low memory complexity algorithm for attacking the model. The algorithm outputs the initialization bits $\mathbf{I}_\mathrm{s}$ for LFSR_s. Knowing $\mathbf{I}_\mathrm{s}$ there exist previous algorithms which can determine $\mathbf{I}_\mathrm{u}$ efficiently.

Acknowledgment

We would like to thank Matthew Parker, John Erik Mathiassen and the anonymous referees for many helpful comments.

References

1. V. Chepyzhov, T. Johansson, and B. Smeets. A simple algorithm for fast correlation attacks on stream ciphers. In *Fast Software Encryption, FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 181–195. Springer-Verlag, 2001.
2. Philippe Chose, Antoine Joux, and Michel Mitton. Fast correlation attacks: An algorithmic point of view. In *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221. Springer-Verlag, 2002.
3. Nicolas Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology-CRYPTO' 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 176–194, 2003.
4. Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359, 2003.
5. J.D Golić. Computation of low-weight parity-check polynomials. *Electronic Letters*, october 1996. 32(21):1981-1982.
6. T. Johansson and F. Jönsson. Theoretical analysis of a correlation attack based on convolutional codes. In *Proceedings of 2000 IEEE International Symposium on Information Theory*, IEEE Transaction on Information Theory, page 212, 2000.
7. Fredrik Jönsson and Thomas Johansson. A fast correlation attack on LILI-128. In *Inf. Process. Lett. 81(3)*, pages 127–132, 2002.
8. Sabine Leveiller, Gilles Zémor, Philippe Guillot, and Joseph Boutros. A new cryptanalytic attack for pn-generators filtered by a boolean function. In *Selected Areas in Cryptography: 9th Annual International Workshop, SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 2003.

9. M. Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology-EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer-Verlag, 1994.
10. W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. In *Advances in Cryptology-EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 301–314. Springer-Verlag, 1998.
11. Håvard Molland. Improved linear consistency attack on irregular clocked keystream generators. In *Fast Software Encryption, FSE 2004*, To appear in LNCS. Springer-Verlag, 2004. Available at http://www.ii.uib.no/~molland/crypto
12. Håvard Molland, John Erik Mathiassen, and Tor Helleseth. Improved fast correlation attack using low rate codes. In *Cryptography and Coding, IMA 2003*, volume 2898 of *Lecture Notes in Computer Science*, pages 67–81, 2003.
13. W.T. Penzhorn and G.J Kuhn. Computation of low-weight parity checks for correlation attacks on stream ciphers. In *Cryptography and Coding, IMA 1995*, volume 1025 of *Lecture Notes in Computer Science*, pages 74–83. Springer-Verlag, 1995.
14. Markku-Juhani Olavi Saarinen. A time-memory tradeoff attack against LILI-128. In *Fast Software Encryption, FSE 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 231–236, 2002.
15. T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Trans. on Comp.*, C-34:81–85, 1985.
16. L. Simpson, E. Dawson, J. Golić, and W. Millan. LILI keystream generator. In *SAC'2000*, volume 2012 of *Lecture Notes in Computer Science*, pages 231–236. Springer-Verlag, 2002. Available at http://www.isrc.qut.edu.au/lili.
17. D. Wagner. A generalized birthday problem. In *Advances in cryptology-CRYPTO' 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303, 2002.
18. Xian-Mo Zhang and Yuliang Zheng. The nonhomomorphicity of boolean functions. In *Selected Areas in Cryptography, SAC 98*, volume 1556 of *Lecture Notes in Computer Science*, pages 280–295. Springer-Verlag, 1998.

Appendix

A Implementation Details

To reach the runtime complexity $O(2^{l_s} \cdot m)$ and memory complexity down to $N + 128m$ bits, the implementation of the algorithm has some tricks. Since not all of these tricks are obvious we give more detailed descriptions of them below.

A.1 Runtime details

Sliding window In Lemma 2 we get $\mathbf{v}^{i+1}$ by among other things deleting the c_0^i first bits of $\mathbf{v}^i$. This is done using the sliding window technique, which means that we move the viewing to the right instead of shifting the whole sequence to the left. This way the shifting can be done in just a couple of operations. To avoid heavy use of memory, we slide the window over an array of fixed length N, so that the entries that become free at the beginning of the array are reused. Thus, the left and right indexes of the sliding window after i iterations will be

$$(left, right) = (i \bmod N, i + N_i \bmod N),$$

where $N > N_i$, for all i, $0 \leq i < 2^{l_s}$.

The same sliding window technique is also used on the equation set when equations are deleted and added to the equation set.

Updating the indices In Lemma 2 every pointer z_{t+1} in $\mathbf{v}^*$ is replaced with z_t for every $0 \leq t \leq M$, which would take M operations. If we skip the replacements we note that after i iterations the entry z_t in $\mathbf{v}^*$ will become z_{t+i}. It is also important to note that when we write $\mathbf{v} = (..., z_0..., z_t, ..., z_M, ...)$, the entries $z_0, ..., z_t, ..., z_M$ are pointers from $\mathbf{v}^*$ to $\mathbf{z}$. They are not the actual key bits. Thus, in the implementation we do not replace z_t with z_{t-1}. But when we after i iterations in the search for equations find an equation $v_k^i + v_{k+j_1}^i + ... + v_{k+j_{w-1}}^i = 0$ that is defined, we replace the corresponding equation $z_{t_1} + z_{t_2} + ... + z_{t_w}$ with $z_{t_1-i} + z_{t_2-i} + ... + z_{t_w-i}$, to compensate.

Reducing the memory access time When we test an equation we must use pointers to pointers to the keystream. Then each equation test will have high memory access time. We can reduce this significantly by testing the equations on 32 states simultaneously. This is possible since the next state $\mathbf{I}_\mathrm{s}^{i+1}$ is tested by shifting all the equations one entry to the left over $\mathbf{z}$. We can now take the bits $z_{t_a}, z_{t_a+1}, ..., z_{t_a+31}$ for each of the term $1 < a \leq 4$ in the equations and put them into 32 bit registers. Now we can test the states and add one to the metrics of the states that satisfy the equation. This speeds up the runtime by a factor of approximately 20.

A.2 Memory Details

Reducing the use of memory Instead of storing all the pointers, we set 1 in $\mathbf{v}^i$ where the bits are defined and 0 otherwise. When we search in $\mathbf{v}^i$ to find entries where the equation λ_v is defined, we keep track of where in $\mathbf{z}$ the four terms in λ_v points to by counting the number of 1's we pass during the search. This is done for each of the 4 terms in the equation λ_v. This way we always know where in $\mathbf{z}$ the given equation of $\mathbf{v}^i$ points to. Using this trick the number of memory bits needed during an attack is reduced from $32N + 128m$ bits to

$$N + 128m$$

Implementing this trick will not affect the runtime of the attack.