Fast Correlation Attacks through Reconstruction of Linear Polynomials

Thomas Johansson and Fredrik Jönsson

Dept. of Information Technology
Lund University, P.O. Box 118, 221 00 Lund, Sweden
{thomas, fredrikj}@it.lth.se

Abstract. The task of a fast correlation attack is to efficiently restore the initial content of a linear feedback shift register in a stream cipher using a detected correlation with the output sequence. We show that by modeling this problem as the problem of learning a binary linear multivariate polynomial, algorithms for polynomial reconstruction with queries can be modified through some general techniques used in fast correlation attacks. The result is a new and efficient way of performing fast correlation attacks.

Keywords. Stream ciphers, correlation attacks, learning theory, reconstruction of polynomials.

1 Introduction

Consider a binary additive stream cipher, i.e., a synchronous stream cipher in which the keystream, the plaintext, and the ciphertext are sequences of binary digits. The output sequence of the keystream generator, $z_1, z_2, \ldots$ is added bitwise to the plaintext sequence $m_1, m_2, \ldots$, producing the ciphertext $c_1, c_2, \ldots$. The keystream generator is initialized through a secret key K, and hence, each key K will correspond to an output sequence. Since the key is shared between the transmitter and the receiver, the receiver can decrypt by adding the output of the keystream generator to the ciphertext and obtain the message sequence, see Figure 1.

The design goal is to efficiently produce random-looking sequences that are as "indistinguishable" as possible from truly random sequences. For a synchronous stream cipher, a known-plaintext attack is equivalent to the problem of finding the key K that produced a given keystream $z_1, z_2, \ldots, z_N$. We assume that a given output sequence from the keystream generator, $z_1, z_2, \ldots, z_N$, is known to the cryptanalyst and that his task is to restore the secret key K.

It is common to use linear feedback shift registers, LFSRs, as building blocks in different ways. Furthermore, the secret key K is usually chosen to be the initial state of the LFSRs. The feedback polynomials of the LFSRs are considered to be known.

Several cryptanalytic attacks against stream ciphers can be found in the literature [14]. One very important class of attacks on LFSR-based stream ciphers

M. Bellare (Ed.): CRYPTO 2000, LNCS 1880, pp. 300–315, 2000.

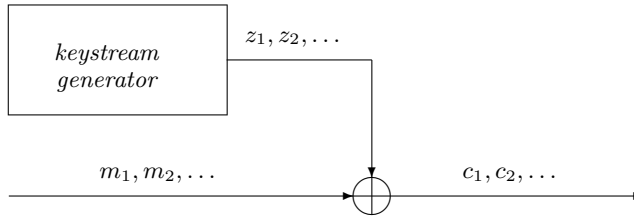

Fig. 1. A binary additive stream ciphers

is *correlation attacks*. The idea is that if one can detect a correlation between the known output sequence and the output of one individual LFSR, it is possible to mount a "divide-and-conquer" attack on the individual LFSR [18,19,12,13], i.e., we try to restore the individual LFSR independently from the other LFSRs. By a correlation we mean that, if $u_1, u_2, \ldots$ denotes the output of the particular LFSR, we have

$$P(u_i = z_i) \neq 1/2, \quad i \geq 1.$$

Other types of correlations may also apply.

A common methodology for producing random-like sequences from LFSRs is to combine the output of several LFSRs by a nonlinear Boolean function f with desired properties [14]. The purpose of f is to destroy the linearity of the LFSR sequences and hence provide the resulting sequence with a large linear complexity [14]. Note that for such a stream cipher, there is always a correlation between the output z_n and either one or a set of output symbols from different LFSRs.

Finding a low complexity algorithm that successfully uses the existing correlations in order to determine a part of the secret key can be a very efficient way of attacking stream ciphers for which a correlation is identified. After the initializing ideas of Siegenthaler [18,19], Meier and Staffelbach [12,13] found a very interesting way to explore the correlation in what was called a *fast* correlation attack. A necessary condition is that the feedback polynomial of the LFSR has a very low weight. This work was followed by several papers, providing improvements to the initial results of Meier and Staffelbach, see [16,4,5,17]. However, the algorithms are efficient (good performance and low complexity) only if the feedback polynomial is of low weight. More recently, steps in other directions were taken, and in [9] it was suggested to use convolutional codes in order to improve performance [9]. This was followed by a generalization in [10], applying the use of iterative decoding and turbo codes. One main advantage compared to previous results was the fact that these algorithms now applied to a feedback polynomial of arbitrary form. Very recently, several other suggested methods have appeared, see [3,15,2].

The purpose of this paper is to show that the initial state recovery problem in a fast correlation attack can be modeled as the problem of learning a binary

linear multivariate polynomial. We show that algorithms for polynomial reconstruction with queries can be modified through some general techniques used in fast correlation attacks. The result is a new and efficient way of performing fast correlation attacks. Actually, two algorithms are presented, one based on a direct search and one based on a sequential procedure. Both provide very good simulation results as well as a theoretical platform.

The paper is organized as follows. In Section 2 we give the preliminaries on the standard model that is used for cryptanalysis and reformulate this into a polynomial reconstruction problem. In Section 3 we review an algorithm by Goldreich, Rubinfeld and Sudan [7] that solves the polynomial reconstruction problem with queries in polynomial time. In Section 4 we derive a new algorithm for fast correlation attacks, inspired by the previous section. In Section 5 we present a sequential version of the new algorithm, i.e, this algorithm builds a tree of possible candidates and searches through it. In Section 6 we present simulation results and a comparison with other algorithms, and in Section 7 a sketch of a theoretical platform for the two algorithms is presented. We show among other things that the central test in the algorithms is statistically optimal.

2 Preliminaries and Model

Most authors [19,12,13,16,4,9,10] use the approach of viewing our cryptanalysis problem as a decoding problem over the binary symmetric channel. However, in this section we show that it can equivalently be viewed as the problem of learning a linear multivariate polynomial.

Let the target LFSR have length l and feedback polynomial $g(x)$. Clearly, the number of possible LFSR sequences is 2^l. Furthermore, assume that the known keystream sequence $\mathbf{z} = z_1, z_2, \ldots, z_N$ is of length N.

The assumed correlation between u_i and z_i is described by the correlation probability $1/2 + \epsilon$, defined by $1/2 + \epsilon = P(u_i = z_i)$, where $0 < \epsilon < 1/2$. The problem of cryptanalysis is the following. Given the received word $(z_1, z_2, \ldots, z_N)$ as the output of the stream cipher, find the initial state (or at least a part of it) of the target LFSR.

It is known that the length N should be at least around $N_0 = l/(1 - h(p))$ for a unique solution to the above problem [4], where $h(p)$ is the binary entropy function. Throughout this paper we assume that $N \gg N_0$. For this case, *fast correlation attacks* are applicable. Although this notation was initially used as the notion for the algorithms developed by Meier and Staffelbach [12,13], we adopt this terminology for any algorithm that finds the correct initial state of the target LFSR significantly faster than exhaustively searching through all initial states.

Let us now consider the unknown initial state of the target LFSR, denoted

$$\mathbf{u} = (u_1, u_2, \ldots, u_l). \tag{1}$$

Clearly, since the LFSR sequence is generated through the recursion

$$u_i = \sum_{j=1}^{l} g_j u_{i-j}, \quad i > l, \tag{2}$$

where $g(x) = 1 + g_1 x + \ldots g_l x^l$, we can express each u_i as some known linear combination of the initial state $\mathbf{u}$, i.e.,

$$u_i = \sum_{j=1}^{l} w_{ij} u_j, \quad \forall i \geq 1, \tag{3}$$

where $w_{ij}, i \geq 1, 1 \leq j \leq l$ are known constants that can be calculated provided $g(x)$ is known (This is essentially the error correcting code one gets by truncating the set of LFSR sequences).

Define the *initial state polynomial*, denoted $U(\mathbf{x})$, to be

$$U(\mathbf{x}) = U(x_1, x_2, \ldots, x_l) = u_1 x_1 + u_2 x_2 + \cdots + u_l x_l. \tag{4}$$

With this notation, we can express each u_i as being the initial state polynomial evaluated in some known point $\mathbf{x}_i = (w_{i1}, w_{i2}, \ldots, w_{ij})$, i.e.,

$$u_i = U(\mathbf{x}_i), \quad i \geq 1. \tag{5}$$

The correlation between u_i and z_i can be described by introducing a noise vector

$$\mathbf{e} = (e_1, e_2, \ldots, e_N), \tag{6}$$

where $e_i \in \mathbb{F}_2$ are independent random variables for $1 \leq i \leq N$ and $P(e_i = 0) = 1/2 + \epsilon$. Then we model the correlation by writing $\mathbf{z} = \mathbf{u} + \mathbf{e}$, giving

$$\mathbf{z} = (U(\mathbf{x}_1) + e_1, U(\mathbf{x}_2) + e_2, \ldots, U(\mathbf{x}_N) + e_N), \tag{7}$$

where $\mathbf{x}_i$ are known l-tuples for all $1 \leq i \leq N$. In conclusion, we have reformulated our problem into the following.

The output vector z consists of a number of noisy observations of an unknown polynomial $U(\mathbf{x})$ evaluated in different known points $\{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$. The task of the attacker is to determine the unknown polynomial $U(\mathbf{x})$.

3 Learning Polynomials with Queries

In computational learning theory (see e.g., [7] and its references), one might want to consider the following general *reconstruction problem:*

Given: An oracle (black box) for an arbitrary unknown function $f : F^l \to F$, a class of functions $\mathcal{F}$ and a parameter δ.

Problem: Provide a list of all functions $g \in \mathcal{F}$ that agree with f on at least a δ fraction of the inputs.

The general reconstruction problem can be interpreted in several ways. We consider only the paradigm of learning with persistent noise. Here we assume that the output of the oracle is derived by evaluating some specific function in $\mathcal{F}$ and then adding noise to the result. A lot of work on different settings for this problem can be found.

We will now pay special attention to the work of Goldreich, Rubinfeld and Sudan in [7]. They consider a case of the reconstruction problem when the hypothesis class $\mathcal{F}$ is the set of linear polynomials in l variables (actually, any polynomial degree d was considered in [7], but we are only interested in the linear case). In the binary case ($F = \mathbb{F}_2$), they demonstrate an algorithm that given $\epsilon > 0$ and provided oracle access to an arbitrary function $f : F^l \to F$, runs in time poly(l/ϵ) and outputs a list of all linear functions in l variables that agree with f on at least $\delta = 1/2 + \epsilon$ of the output.

Let us immediately describe the procedure. First, the problem description can be as follows. On a selected input $\mathbf{x}$, the oracle evaluates an unknown linear function $p(\mathbf{x})$, adds a noise value e, and outputs the result $p(\mathbf{x}) + e$. On the next oracle access, the function is evaluated in a new point and a new noise value is added.

The algorithm for solving the above problem given in [7] is a generalization of an algorithm given in [6] (in the binary case that we consider they coincide). Consider all polynomials of the form

$$p(\mathbf{x}) = \sum_{i=1}^{l} c_i x_i.$$

The algorithm uses the concept of *i-prefixes*, which is defined to be all polynomials that can be expressed in the form $p(x_1, x_2, \ldots, x_i, 0, 0, \ldots, 0)$. This means that an i-prefix is a polynomial in l variables in which only the first i variables appear.

The algorithm proceeds in l rounds, so that in the ith round we have a list of candidates for the i-prefixes of $p(\mathbf{x})$. The list of i-prefixes is generated by extending the list of $(i-1)$-prefixes from the previous round in all possible ways, i.e., by adding or not adding the x_i variable to each of the members of the $(i-1)$-prefixes. Hence the list is doubled in cardinality. After the extension, a screening process takes place. The screening process guarantees that the i-prefix of the correct solution passes with high probability and that not too many other prefixes pass.

The screening process is done by testing each candidate prefix, denoted $(c_1, c_2, \ldots, c_i)$, as follows. Pick $n = \text{poly}(l/\epsilon)$ sequences uniformly from $\mathbb{F}_2^{l-i}$. For each such sequence, denoted $(s_{i+1}, \ldots, s_l)$, and for every $\xi \in \mathbb{F}_2$, estimate the quantity

$$P(\xi) = Pr_{r_1,\ldots,r_i \in \mathbb{F}_2} \left[f(\mathbf{r}, \mathbf{s}) = \sum_{j=1}^{i} c_j r_j + \xi \right].$$

Here $(\mathbf{r}, \mathbf{s})$ denotes the vector $(r_1, \ldots, r_i, s_{i+1}, \ldots, s_l)$. All these probabilities can be approximated simultaneously by using a sample of poly(l/ϵ) sequences

$(r_1, \ldots, r_i)$. A candidate is considered to pass the test if for at least one sequence $(s_{i+1}, \ldots, s_l)$ there exists ξ such that the estimate $P(\xi)$ is greater than $1/2+\epsilon/3$. It is shown in [7] that the correct candidate passes the test with overwhelming probability, and that not too many other candidates do. For more details on this algorithm, we refer to [7].

4 Fast Correlation Attacks Based on Algorithms for Learning Polynomials

We observe the similarities between our correlation attack problem described as a polynomial reconstruction problem as in Section 2, and the problem of learning polynomials with queries as described in the previous section.

Note that the polynomial time algorithm of the previous section can not be applied directly to the correlation attack problem, since queries are essential. *In the query case, sample points given to the oracle can be chosen, whereas for correlation attacks the sample points are simply randomly selected.* The latter problem is actually a well-known problem also in learning theory, called "learning parity with noise", and it is commonly believed to be hard, see [11,1].

Nevertheless, we are interested in finding as efficient correlation attacks as possible, and we will now derive an algorithm that is inspired by the results presented in the previous section.

Let us first briefly review our problem formulation. The recovery of the initial state of the target LFSR is viewed as the problem of recovering an unknown binary linear polynomial $U(\mathbf{x})$ in l variables. To our disposal, we have a number N of noisy observations of this polynomial (the output sequence), denoted

$$\mathbf{z} = (z_1, z_2, \ldots, z_N).$$

The noise is such that

$$P(z_i = U(\mathbf{x}_i)) = 1/2 + \epsilon, \quad 1 \leq i \leq N,$$

where $\mathbf{x}_i$ are known random l-tuples for all $1 \leq i \leq N$.

Our problem in applying the algorithm described in Section 3 is the fact that we are not able to select the points $\mathbf{x}_i$ ourselves. This can to some extent be compensated for by the following observation [9].

Assume that we have noisy observations z_i and z_j of the polynomial $U(\mathbf{x})$ in two points $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively, i.e., $P(z_i = U(\mathbf{x}_i)) = 1/2 + \epsilon$ and $P(z_j = U(\mathbf{x}_j)) = 1/2 + \epsilon$. Since $U(\mathbf{x})$ is a linear polynomial, the sum of these two noisy observations will give rise to an even more noisy observation in the point $\mathbf{x}_i + \mathbf{x}_j$, since

$$\begin{aligned} P(z_i + z_j = U(\mathbf{x}_i + \mathbf{x}_j)) &= P(z_i + z_j = U(\mathbf{x}_i) + U(\mathbf{x}_j)) \\ &= P(z_i = U(\mathbf{x}_i))P(z_j = U(\mathbf{x}_j)) \\ &\quad + P(z_i \neq U(\mathbf{x}_i))P(z_j \neq U(\mathbf{x}_j)) \\ &= (1/2 + \epsilon)^2 + (1/2 - \epsilon)^2 \\ &= 1/2 + 2\epsilon^2. \end{aligned}$$

Next, observe that we do not have to restrict ourselves to addition of just two sample points, but can consider any sum of t points. Hence, any $\sum_{j=1}^{t} z_{a_j}$, $a_1, \ldots a_t \in \{1, 2, \ldots, N\}$, will be a noisy observation of $U(\sum_{j=1}^{t} \mathbf{x}_{a_j})$ with noise level

$$P(\sum_{j=1}^{t} z_{a_j} = U(\sum_{j=1}^{t} \mathbf{x}_{a_j})) = 1/2 + 2^{t-1}\epsilon^t. \tag{8}$$

For convenience, we introduce the notation $\hat{\mathbf{x}} = \sum_{j=1}^{t} \mathbf{x}_{a_j}$ and $\hat{z} = \sum_{j=1}^{t} z_{a_j}$ and write

$$U(\hat{\mathbf{x}}) = \hat{z} + e,$$

where now e is a binary random variable with $P(e = 0) = 1/2 + 2^{t-1}\epsilon^t$, from (8).

If we want to use the algorithm in Section 3 we must feed the oracle with $\hat{\mathbf{x}}$ points of a special form. An idea in the algorithm to be described is to construct such points by adding suitable vectors $\mathbf{x}_{a_j}$ in such a way that their sum is of the required form. Clearly, the noise level increases with the number of vectors in the sum, so we are interested in having as few vectors as possible summing to the desired form. On the other hand, allowing only very few vectors in the sum will give us only very few $\hat{\mathbf{x}}$ vectors of the desired form. Hence, there is a tradeoff for the value of the constant t. We return to this issue in the theoretical analysis.

Also, we introduce a slightly modified version of the algorithm from Section 3. The new version includes a squared distance used in the test in the screening procedure. We will later show that this is a statistically optimal distance measure. We first consider a version in which the idea of i-prefixes is removed. In the next section we elaborate on the idea of i-prefixes. A description of the basic algorithm is given in Figure 2.

Let us give an intuitive explanation of the algorithm. We first note that the algorithm recovers the first k bits of the initial state, namely $u_1, \ldots, u_k$. The remaining part of the initial state can be recovered in a similar way, if desired.

Now consider the case of one hypothesized value of $(u_1, \ldots, u_k)$. We want to check whether this value, denoted $(\hat{u}_1, \ldots, \hat{u}_k)$, is correct or not. This is done by first selecting a certain $(l-k)$-tuple $\mathbf{s}_i$, and then by finding all linear combinations of t vectors in $\{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$,

$$\hat{\mathbf{x}}(i) = \sum_{j=1}^{t} \mathbf{x}_{a_j}, \tag{9}$$

having the special form

$$\hat{\mathbf{x}}(i) = (\hat{x}_1, \ldots, \hat{x}_k, \mathbf{s}_i), \tag{10}$$

for arbitrary values of $\hat{x}_1, \ldots, \hat{x}_k$ (not all zero). The complexity of this precomputation step depends on t, and by using some simple birthday-paradox arguments, one can show that the computation can be done in $O(N^{\lceil t/2 \rceil})$ using $O(N^{\lfloor t/2 \rfloor})$ storage.

In: $\mathbf{z} = (z_1, \ldots, z_N)$, $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$, and constants t, k and n.

1. (Precomputation) Select n different $(l-k)$-tuples $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n$. For each $\mathbf{s}_i$, find all linear combinations of the form $\hat{\mathbf{x}}(i) = \sum_{j=1}^{t} \mathbf{x}_{a_j}$ which are of the special form

$$\hat{\mathbf{x}}(i) = (\hat{x}_1, \ldots, \hat{x}_k, \mathbf{s}_i),$$

for arbitrary values of $\hat{x}_1, \ldots, \hat{x}_k$.
Store all $\hat{\mathbf{x}}(i)$ together with all $\hat{z}(i) = \sum_{j=1}^{t} z_{a_j}$. Let the set of all such pairs have cardinality S_i.
2. Run through all 2^k values of the constants $(u_1, \ldots, u_k) = (\hat{u}_1, \ldots, \hat{u}_k)$ as follows.
3. For each s_i, run through all S_i stored pairs $\{(\hat{\mathbf{x}}(i), \hat{z}(i))\}$, calculate the number of times we have

$$\sum_{j=1}^{k} \hat{u}_j \hat{x}_j = \hat{z}(i),$$

and denote this by *num*. Update

$$dist \leftarrow dist + (S_i - 2 \cdot num)^2.$$

4. If *dist* is the highest received value so far, store $(\hat{u}_1, \ldots, \hat{u}_k)$. Set $dist \leftarrow 0$.

Out: Output $(\hat{u}_1, \ldots, \hat{u}_k)$ having the highest value of *dist*.

Fig. 2. A description of the basic algorithm

Now, the main observation is that the relation between $U(\hat{\mathbf{x}}(i))$ and $\hat{z}(i)$ can, from our previous arguments, be written in the form

$$U(\hat{\mathbf{x}}(i)) = \hat{z}(i) + e, \tag{11}$$

where e represents the noise having a noise level of $P(e = 0) = 1/2 + 2^{t-1}\epsilon^t$. Now (11) is equivalently expressed as

$$\sum_{j=1}^{k} u_j \hat{x}_j + \sum_{j=k+1}^{l} u_j s_j = \hat{z}(i) + e, \tag{12}$$

and this can be rewritten as

$$\sum_{j=1}^{k} (u_j + \hat{u}_j)\hat{x}_j + \sum_{j=k+1}^{l} u_j s_j + e = \sum_{j=1}^{k} \hat{u}_j \hat{x}_j + \hat{z}(i). \tag{13}$$

Now recall that $W = \sum_{j=k+1}^{l} u_j s_j$ in (13) is a fixed binary random variable for all linear combinations of the special form that we required, i.e., we will have either $W = 0$ for all our $\hat{\mathbf{x}}(i)$'s, or $W = 1$.

Consider a correct hypothesized value and assume that we have all the S_i equations. Then *num* simply counts the number of times the right hand side in (13) is zero. Since $\sum_{j=1}^{k}(u_j + \hat{u}_j)\hat{x}_j = 0$, the probability for the left hand side to be zero is then $P(W + e = 0)$. This probability is either $1/2 - 2^{t-1}\epsilon^t$ or $1/2 + 2^{t-1}\epsilon^t t$ for all equations, depending on whether $W = 0$ or $W = 1$. Thus *num* has a binomial distribution $Bin(S_i, p)$, with p being one of the two probabilities above.

However, if the hypothesized valued was wrong, then $\sum_{j=1}^{k}(u_j + \hat{u}_j)\hat{x}_j \neq 0$, and hence, it will result in *num* being binomial distributed, $Bin(S_i, p)$, with $p = 1/2$.

In order to separate the two hypothesis we measure the difference between the number of times $\sum_{j=1}^{k} \hat{u}_j\hat{x}_j = \hat{z}(i)$ holds and the number of times it does not hold. Then a squared distance $((S_i - 2 \cdot num)^2)$ is used. If we have enough points, i.e., we can create enough different $\hat{\mathbf{x}}$ as linear combinations of at most t $\mathbf{x}_i$'s, we will be able to separate the two hypotheses. However, the number of linear combinations for a particular $\mathbf{s}_i$ value is limited. Hence, we also run through a lot of different $\mathbf{s}_i$ values. Each gives a squared distance, and we sum them all up to become our overall distance (*dist*). In Section 7 we show that a squared distance leads to a statistically optimal test, i.e., we pick the candidate having the highest probability.

5 A Sequential Reconstruction Algorithm

In this section we want to elaborate around the idea of using i-prefixes from Section 3 and modify the proposed algorithm into a sequential algorithm.

Instead of simply selecting the candidate $(\hat{u}_1, \ldots, \hat{u}_k)$ having the highest value of *dist*, we would now want to have a set of surviving candidates. These are then extended by incrementing, in our case, k by one. This extension doubles the number of candidates, since each surviving candidate can be extended in the $(k + 1)$th position by either 0 or 1. But before the next extension, we run a screening procedure that removes a substantial part of the candidates.

This is a straightforward usage of the idea of i-prefixes. From our perspective, it does introduce some small practical problems. The major problem is that we now must store a large set of possible candidates. The performance of our algorithms is highly connected with the computational complexity. If a large memory must be used in our algorithm, some degradation in complexity is likely to appear in practice. This is the reason for presenting a slightly different approach. Essentially we use, instead of an l round algorithm, a tree structure for all candidates that are still "alive". The advantage is that, essentially, the memory requirements are removed. Figure 3 shows how a version of such an algorithm may look like.

Note that the set Ω is only introduced to simplify the presentation. We do not need to store it. It is a lexicographically ordered set, and when we put new values in Ω, we actually do not need to store anything.

In: $\mathbf{z} = (z_1, \ldots, z_N)$, $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$, and constants t, $\hat{k}$, n and $threshold(k)$. Let Ω be a list of all $\hat{k}$-tuples in lexicographical order.

1. (Precomputation) For each value of k, $\hat{k} \leq k \leq l$, set up a screening procedure as given in 2.
2. (Precomputation) Select n different $(l-k)$-tuples $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n$. For each $\mathbf{s}_i$, find all linear combinations of the form $\hat{\mathbf{x}}(i) = \sum_{j=1}^{t} \mathbf{x}_{a_j}$ which are of the special form

$$\hat{\mathbf{x}}(i) = (\hat{x}_1, \ldots, \hat{x}_k, \mathbf{s}_i),$$

for arbitrary values of $\hat{x}_1, \ldots, \hat{x}_k$. Store $(\hat{\mathbf{x}}(i), \hat{z}(i) = \sum_{j=1}^{t} z_{a_j})$. Assume that S_i such pairs have been stored.
3. Take the first value in Ω, denoted $(\hat{u}_1, \ldots, \hat{u}_k)$.
4. For each $\mathbf{s}_i$, run through all S_i stored pairs for $\mathbf{s}_i$, calculate the number of times we have

$$\sum_{j=1}^{k} \hat{u}_j \hat{x}_j = \hat{z}(i),$$

and denote this by *num*. Update

$$dist \leftarrow dist + (S_i - 2 \cdot num)^2.$$

5. If $dist > threshold(k)$, put both $(\hat{u}_1, \ldots, \hat{u}_k, 0)$ and $(\hat{u}_1, \ldots, \hat{u}_k, 1)$ in Ω. Set $dist \leftarrow 0$. If $|\Omega| > 1$ go to 3.

Out: Output all values in Ω that has reached length l.

Fig. 3. A description of the sequential algorithm.

Example 1. Assume that the sequential algorithm is applied with $\hat{k} = 5$. We examine first the value $(u_1, \ldots, u_5) = (0, 0, 0, 0, 0)$. Assume that the received *dist* is higher than $threshold(5)$. We then extend this vector with the two possible values for u_6, giving $(0, 0, 0, 0, 0, 0)$ and $(0, 0, 0, 0, 0, 1)$. We continue to examine the first of these candidates. Assume that $dist < threshold(6)$. We continue with the second of these candidates. Assume that in this case $dist > threshold(6)$. We extend this vector and get $(0, 0, 0, 0, 0, 1, 0)$ and $(0, 0, 0, 0, 0, 1, 1)$ as two new vectors. We continue in this fashion. The tree structure of this procedure is presented in Figure 4.

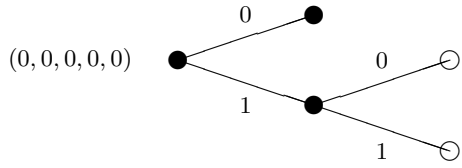

Fig. 4. The tree in Example 1.

Whether a candidate will survive the test at level k or not is determined by a threshold value ($threshold(k)$). Increasing the threshold value will throw away more wrong candidates, but will also increase the probability of throwing away the correct candidate. A discussion on how to choose the threshold values is given in Section 7.

Comparing with the algorithm of the previous section, this algorithm will have a better performance (fewer tests on average) if we implement it in an efficient way. One important observation in this direction is the fact that all $\hat{\mathbf{x}}(i)$ vectors for a certain k will appear again as valid $\hat{\mathbf{x}}(i)$ vectors for higher k (assuming that we use the same $\mathbf{s}_i$ vectors). This means that we should not recalculate num on $\hat{\mathbf{x}}(i)$ vectors that have already been used, but rather, we store the value of num for all $\mathbf{s}_i$ values and incorporate this in the calculation for higher k values.

6 Performance of the Proposed Algorithm

In this section we present some simulation results for the basic algorithm described in Section 4 and the sequential version in Section 5. Simulations are presented for $t = 2$ and $t = 3$. In general, increasing t will increase the performance at the cost of an increased precomputation time and increased memory requirement in precomputation.

The first simulations are for the same feedback polynomial and the same length of the observed keystream as in [9,10,2]. Table 1 shows the maximum error probability $p = 1/2 - \epsilon$ for the basic algorithm when the received sequence is of length $N = 400000$. The parameter k is varying in the range $13-16$ and n is in the set $n \in \{1, 2, 4, 8, \ldots, 512\}$. As a particular example, when $k = 16$, $n = 256$

Table 1. Maximum $p = 1/2 - \epsilon$ for the basic algorithm with $t = 2$, $k = 13, \ldots, 16$, varying n, and $N = 400000$.

	$N = 400000$			
n	$k = 13$	$k = 14$	$k = 15$	$k = 16$
1	0.30	0.32	0.34	0.36
2	0.32	0.34	0.36	0.38
4	0.34	0.36	0.38	0.40
8	0.36	0.38	0.40	0.41
16	0.38	0.39	0.41	0.42
32	0.39	0.40	0.42	0.43
64	0.40	0.41	0.42	0.44
128	0.41	0.42	0.43	0.44
256	0.42	0.43	0.43	0.45
512	0.42	0.44	0.44	0.45

we reach $p = 0.45$ having 400000 known keystream symbols. The running time is less than 3 minutes, and the precomputation time negligible.

It is important to observe that for a fixed running time, the performance increases with increasing n (up to a certain point). The table entries $\{k = 16, n = 1\}$, $\{k = 15, n = 4\}$, $\{k = 14, n = 16\}$, $\{k = 13, n = 64\}$ all have roughly the same computational complexity, but an increasing performance with n can be observed.

More interesting is perhaps to show simulation results for longer LFSRs, as was done in [3]. We present results for the basic algorithm when $l = 60$ using a feedback polynomial of weight 13. In Table 2 we show the required

$l = 60$, $t = 2$

N	k	n	time	p
$40 \cdot 10^6$	23	1	96 sec	0.35
$40 \cdot 10^6$	22	2	48 sec	0.36
$40 \cdot 10^6$	21	4	25 sec	0.36
$40 \cdot 10^6$	25	1	26 min	0.40
$40 \cdot 10^6$	24	2	13 min	0.40
$40 \cdot 10^6$	23	4	6.5 min	0.40
$40 \cdot 10^6$	22	8	3.3 min	0.41
$40 \cdot 10^6$	25	4	106 min	0.43

$l = 60$, $t = 3$

N	k	n	time	p
$1.5 \cdot 10^5$	24	1	4.5 min	0.3
$1.5 \cdot 10^5$	23	2	2.3 min	0.3
$1.5 \cdot 10^5$	22	4	69 sec	0.3
$1.5 \cdot 10^5$	25	1	18 min	0.32
$1.5 \cdot 10^5$	24	2	9.2 min	0.32
$1.5 \cdot 10^5$	23	4	4.6 min	0.32

Table 2. Performance of the basic algorithm with $l = 60$ when $t = 2$ and $t = 3$, respectively.

computational complexity and the achieved correlation probability for different algorithm parameters. The implementations were written in C and the running times were measured on a Sun Ultra-80 running under Solaris.

We can compare with other suggested methods. Actually, in the special case of $n = 1$, our proposed algorithm will coincide with the method in [3]. This enables us to see the improvement in Table 2, by observing the decrease of decoding time when n increases (for a fixed p). Furthermore, [3] is the only previous work reporting simulation results for $l \geq 60$.

An important advantage for the proposed methods is the storage complexity. The attacks based on convolutional and turbo codes [9,10] uses a trellis with 2^B states. Hence, the size of B is limited to $20 - 30$ in practise, due to the fact that it must be kept in memory during decoding. On the other hand, the memory requirements for the algorithms presented in this paper remain constant when k increases. Also, the proposed algorithms are trivially parallelizable, and hence the only limiting factor is the total computational complexity.

Finally, simulation results for the sequential algorithm should be considered. Some initial simulations for the case $N = 400000$, $p = 0.40$, $t = 2$, $n = 64$ indicated a speedup factor of approximately 5. An extensive set of simulations for the sequential algorithm is under progress.

7 Theoretical Analysis of the Algorithms

In this section we sketch some results for a theoretical analysis of the proposed algorithms. A complete analysis will appear in the full paper. Here we prove, among other things, that using the squared distance is statistically optimal.

First, we derive an expression for the expected number of linear combinations of the form (9) and (10), i.e., the expected value of the parameter S_i in the algorithm.

Lemma 1. *Let $E[S]$ be the expected number of linear combinations of the form (9) and (10) that can be created from t out of N random vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$. Then $E[S]$ is given as*

$$E[S] = \frac{\binom{N}{t}}{2^{l-k}}.$$

Proof: The number of ways we can create a linear combinations of of the form (9) is $\binom{N}{t.}$ The probability of getting a particular value of $\mathbf{s}$ is $1/2^{l-k}$. Thus, we have in average $\binom{N}{t}/2^{l-k}$ linear combinations ending with a particular value $\mathbf{s}$. ■

Next we show that using *dist* as defined in Section 4 gives optimal performance. We start by considering the case when we have one fixed value of $\mathbf{s}$. Then we generalize to the case with several different $\mathbf{s}$ vectors.

Assume that for the given $\mathbf{s}$ we have created S noisy observations of the polynomial $U(\mathbf{x})$. The expected value of S is then given by Lemma 1. Assume further that we are considering a particular candidate $(\hat{u}_1, \ldots, \hat{u}_k)$, and that we have found *num* observations such that $\sum_{j=1}^{k} \hat{u}_j \hat{x}_j = \hat{z}(i)$. Consider two hypothesis H_0, and H_1. Let H_1 be the hypothesis that the candidate is correct, and H_0 that the candidate is wrong.

Introduce the random variable $W = \sum_{j=k+1}^{l} u_j s_j$. Define p_0 as $p_0 = P(e = 0) = 1/2 + 2^{t-1}\epsilon^t$. Furthermore, $P(W = 0) = 1/2$. We showed in Section 4 the following distribution for *num*:

$$\begin{aligned} &num|H_0 \in Bin(S, 1/2), \\ &num|H_1, W = 0 \in Bin(S, p_0), \\ &num|H_1, W = 1 \in Bin(S, (1 - p_0)). \end{aligned}$$

Next, we approximate the binomial distribution for *num* with a normal distribution. As long as, $Spq \gg 10$ the approximation will be good. Furthermore, we define the random variable Y as $Y = |2 \cdot num - S|$. If $P((2 \cdot num - S) < 0|H_1, W = 0) = P((2 \cdot num - S) > 0|H_1, W = 1)$ is small then we get the following distribution of Y:

$$\begin{aligned} f_{Y|H_0}(y) &= \frac{2}{\sqrt{\pi S}} e^{-y^2/S} \\ f_{Y|H_1}(y) &= \frac{1}{\sqrt{4\pi S p_0(1-p_0)}} e^{-\frac{(y - Sp_0)^2}{4Sp_0(1-p_0)}} \end{aligned}$$

The estimate of $(u_1, \ldots, u_k)$ is taken as the $(\hat{u}_1, \ldots, \hat{u}_k)$ for which $P(H_1|Y)$ is maximal. However, it is not possible to calculate $P(H_1|Y)$ directly. Instead, we can equivalently choose the estimate as the $(\hat{u}_1, \ldots, \hat{u}_k)$ for which the likelihood ratio

$$\Lambda = \frac{P(H_1|Y)}{1 - P(H_1|Y)} = \frac{P(H_1|Y)}{P(H_0|Y)} = \frac{P(Y|H_1)P(H_1)}{P(Y|H_0)P(H_0)},$$

is maximal.

In our case it is more convenient to use the loglikelihood ratio $\lambda = \ln(\Lambda)$. Thus we can formulate the problem of finding the most likely candidate as:

$$\arg \max_{(\hat{u}_1,\ldots,\hat{u}_k)} \left[\ln P(Y|H_1) + \ln P(H_1) - \ln P(Y|H_0) - \ln P(H_0)\right].$$

It now follows that maximizing λ is equivalent to taking the candidate for which y^2 is maximum.

This derivation holds when we have one value of S. Now we assume that we have n different S_i, $(S_1, S_2, \ldots, S_n)$ and corresponding $Y = (Y_1, Y_2, \ldots, Y_n)$. By observing that

$$P(Y|H_0) = P(Y_1|H_0)P(Y_2|H_0)\cdots P(Y_n|H_0)$$

we see that optimality is reached for $dist = dist_1 + dist_2 + \ldots + dist_n$, where $dist_i = y_i^2$. In conclusion, we have showed that the chosen distance is statistically optimal.

To analyze the performance of the algorithm when we use the quadratic distance measure we use the following approach. Assume that we have the correct candidate $(\hat{u}_1, \ldots, \hat{u}_k)$. Then we have

$$|2 \cdot num_i - S_i| \in N(S_i(2p_0 - 1), 2S_i p_0(1 - p_0)).$$

If we instead assume that the candidate $(\hat{u}_1, \ldots, \hat{u}_k)$ is wrong we get

$$(2 \cdot num_i - S_i) \in N(0, S/2).$$

The value of $dist$ is calculated as

$$dist = \sum_{i=1}^{n} (2 \cdot num_i - S_i)^2 = \sum_{i=1}^{n} |2 \cdot num_i - S_i|^2.$$

One sees that $dist$ is calculated by squaring and adding n normal random variables. Hence, $dist$ is a noncentral chi-square distributed random variable with n degrees of freedom.

By using the central limit theorem, we get that for large values of n we will be successful when $E(dist|H_1) > E(dist|H_0)$. Since this inequality always holds, the conclusion is that if $n \to \infty$ we can have $\epsilon \to 0$.

Finally we consider the sequential algorithm of Section 5. When analyzing this algorithm we are now interested in two properties. The first is the probability that we accept a candidate as correct when it actually is wrong, denoted by P_F,

(false alarm). The second is the probability that we do not accept a correct candidate. Denote this by P_M, (miss).

Since we know the distribution of *dist* under the hypothesis H_0 and H_1 we can calculate P_M and P_F as follows. Consider a fixed threshold T. The probability of miss P_M is then given as

$$P_M = P(dist < T|H_1),$$

and in the same way we get

$$P_F = P(dist > T|H_0).$$

8 Conclusions

In this work we have shown how learning theory can be used as a basis for correlation attacks on stream ciphers. Techniques for reconstructing polynomials have been modified and combined with some general techniques from correlation attacks. The performance has been demonstrated through a sketch of a theoretical analysis as well as through simulations. The simulations show a very good performance.

The problem that arises in a standard correlation attack is equivalent to the problem of learning parity with noise, a well known problem in computational learning theory, commonly believed to be a hard problem. This might indicate that it is hard to find further significant improvements on the problem. One interesting idea would be to examine whether recent results on polynomial reconstruction as a decoding tool for certain error correcting codes [20] can be used. Some results in this direction can be found in [8].

Acknowledgement

The authors are supported by the Foundation for Strategic Research - PCC under Grant 9706-09.

References

1. A. Blum, M. Furst, M. Kearns, R. Lipton, "Cryptographic primitives based on hard learning problems", *Advances in Cryptology–CRYPTO'93*, Lecture Notes in Computer Science, vol. 773, Springer-Verlag, 1993, pp. 278–291.
2. A. Canteaut, M. Trabbia, "Improved fast correlation attacks using parity-check equations of weight 4 and 5", *Advances in Cryptology–EUROCRYPT'2000*, Lecture Notes in Computer Science, vol. 1807, Springer-Verlag, 2000, pp. 573–588.
3. V. Chepyzhov, T. Johansson, and B. Smeets, "A simple algorithm for fast correlation attacks on stream ciphers", *Fast Software Encryption, FSE'2000*, to appear in Lecture Notes in Computer Science, Springer-Verlag, 2000.
4. V. Chepyzhov, and B. Smeets, "On a fast correlation attack on certain stream ciphers", In *Advances in Cryptology–EUROCRYPT'91*, Lecture Notes in Computer Science, vol. 547, Springer-Verlag, 1991, pp. 176–185.

5. A. Clark, J. Golic, E. Dawson, "A comparison of fast correlation attacks", *Fast Software Encryption, FSE'96*, Lecture Notes in Computer Science, Springer-Verlag, vol. 1039, 1996, pp. 145–158.
6. O. Goldreich and L.A. Levin, "A hard-core predicate for all one-way functions", *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, Seattle, Washington, 15-17 May 1989, pp. 25–32.
7. O. Goldreich, R. Rubinfeld, M. Sudan, "Learning polynomials with queries: The highly noisy case", *36th Annual Symposium on Foundation of Computer Science*, Milwaukee, Wisconsin, 23-25 October 1995, pp. 294–303.
8. T. Jakobsen, "Higher-Order Cryptanalysis of Block ciphers", Ph.D Thesis, Technical University of Denmark, 1999.
9. T. Johansson, F. Jönsson, "Improved fast correlation attacks on stream ciphers via convolutional codes", *Advances in Cryptology–EUROCRYPT'99*, Lecture Notes in Computer Science, vol. 1592, Springer-Verlag, 1999, pp. 347–362.
10. T. Johansson, F. Jönsson, "Fast correlation attacks based on turbo code techniques", *Advances in Cryptology–CRYPTO'99*, Lecture Notes in Computer Science, vol. 1666, Springer-Verlag, 1999, pp. 181–197.
11. M. Kearns, "Efficient noise-tolerant learning from statistical queries", *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, San Diego, California, 16-18 May 1993, pp. 392–401.
12. W. Meier, and O. Staffelbach, "Fast correlation attacks on stream ciphers", *Advances in Cryptology–EUROCRYPT'88*, Lecture Notes in Computer Science, vol. 330, Springer-Verlag, 1988, pp. 301–314.
13. W. Meier, and O. Staffelbach, "Fast correlation attacks on certain stream ciphers", *Journal of Cryptology*, vol. 1, 1989, pp. 159–176.
14. A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
15. M. Mihaljevic, M. Fossorier, and H. Imai, "A low-complexity and high-performance algorithm for the fast correlation attack", *Fast Software Encryption, FSE'2000*, to appear in Lecture Notes in Computer Science, Springer-Verlag, 2000.
16. M. Mihaljevic, and J. Golic, "A fast iterative algorithm for a shift register initial state reconstruction given the noisy output sequence", *Advances in Cryptology–AUSCRYPT'90*, Lecture Notes in Computer Science, vol. 453, Springer-Verlag, 1990, pp. 165-175.
17. W. Penzhorn, "Correlation attacks on stream ciphers: Computing low weight parity checks based on error correcting codes", *Fast Software Encryption, FSE'96*, Lecture Notes in Computer Science, vol. 1039, Springer-Verlag, 1996, pp. 159–172.
18. T. Siegenthaler, "Correlation-immunity of nonlinear combining functions for cryptographic applications", *IEEE Trans. on Information Theory*, vol. IT–30, 1984, pp. 776–780.
19. T. Siegenthaler, "Decrypting a class of stream ciphers using ciphertext only", *IEEE Trans. on Computers,* vol. C–34, 1985, pp. 81–85.
20. M. Sudan, "Decoding of Reed Solomon codes beyond the error-correction bound", *Journal of Complexity*, vol. 13(1), March 1997, pp. 180–193.