Improved Fast Correlation Attacks on Stream Ciphers via Convolutional Codes*

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. This paper describes new methods for fast correlation attacks, based on the theory of convolutional codes. They can be applied to arbitrary LFSR feedback polynomials, in opposite to the previous methods, which mainly focus on feedback polynomials of low weight. The results improve significantly the few previous results for this general case, and are in many cases comparable with corresponding results for low weight feedback polynomials.
Keywords: Stream ciphers, Correlation attacks, Convolutional codes.

1 Introduction

A binary additive stream cipher is a synchronous stream cipher in which the keystream, the plaintext and the ciphertext are sequences of binary digits. The output of the keystream generator, $z_1, z_2, \ldots$ is added bitwise to the plaintext sequence $m_1, m_2, \ldots$, producing the ciphertext $c_1, c_2, \ldots$. Each secret key k as input to the keystream generator corresponds to an output sequence. Since the secret key k is shared between the transmitter and the receiver, the receiver can decrypt, and obtain the message sequence, by adding the output of the keystream generator to the ciphertext, see Figure 1.

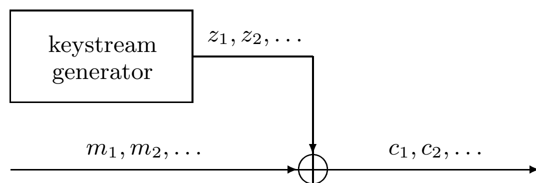

Fig. 1. Principle of binary additive stream ciphers

* This work was supported by the Foundation for Strategic Research - PCC under Grant 9706-09.

J. Stern (Ed.): EUROCRYPT'99, LNCS 1592, pp. 347–362, 1999.
© Springer-Verlag Berlin Heidelberg 1999

The goal in stream cipher design is to efficiently produce random-looking sequences that in some sense are "indistinguishable" from truly random sequences. From a cryptanalysis point of view, a good stream cipher should be resistant against a *known-plaintext attack*. In a known-plaintext attack the cryptanalyst is given a plaintext and the corresponding ciphertext, and the task is to determine a key k. For a synchronous stream cipher, this is equivalent to the problem of finding the key k that produced a given keystream $z_1, z_2, \ldots, z_N$. Throughout this paper, we hence assume that a given keystream $z_1, z_2, \ldots, z_N$ is in the cryptanalyst's possession and that cryptanalysis is the problem of restoring the secret key.

In stream cipher design, one usually use linear feedback shift registers, LFSRs, as building blocks in different ways, and the secret key k is often chosen to be the initial state of the LFSRs.

There are several classes of general cryptanalytic attacks against stream ciphers [9]. In our opinion, the most important class of attacks on LFSR-based stream ciphers is *correlation attacks*. Basically, if one can in some way detect a correlation between the known output sequence and the output of one individual LFSR, this can be used in a "divide-and-conquer" attack on the individual LFSR [12,13,7,8]. There is no requirement of structure of any kind for the key generator. The only thing that matters is the fact that, if $u_1, u_2, \ldots$ denotes the output of the particular LFSR, we have a correlation of the form $P(u_i = z_i) \neq 0.5$, see Figure 2.

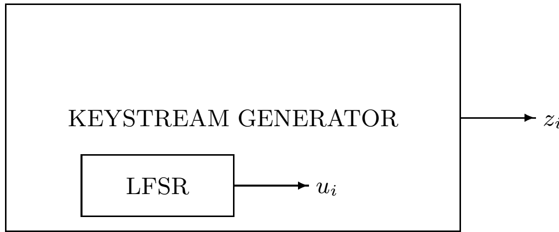

Fig. 2. A sufficient requirement for a correlation attack, $P(u_i = z_i) \neq 0.5$.

A "textbook" methodology for producing random-like sequences from LFSRs is to combine the output of several LFSRs by a nonlinear function f with desired properties. Here f is a binary boolean function in n variables. The purpose is to destroy the linearity of the LFSR sequences and hence provide the resulting sequence with a large linear complexity [9]. This is depicted in Figure 3.

It is worth noticing that there always exists a correlation between the output z_i and either one or a set of M LFSR output symbols $\{u_i^{(i_1)}, u_i^{(i_2)}, \ldots, u_i^{(i_M)}\}$ in the model above. It is well known that if f is a $(M-1)$-resilient (but not M-resilient) function then there is a correlation which can be expressed in the

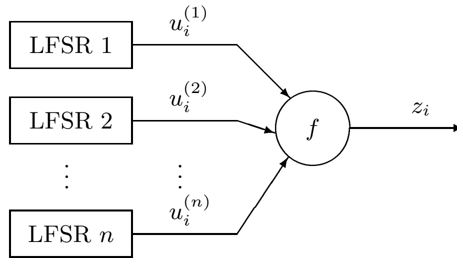

Fig. 3. Principle of nonlinear combination generators

form $P(z_i = u_i^{(i_1)} + u_i^{(i_2)} + \cdots + u_i^{(i_M)}) \neq 0.5$. It is also known that there is a tradeoff between the resiliency and the nonlinearity of f, and hence M must be rather small [12].

Returning to the previously mentioned correlation attacks, the above overview demonstrates that finding a low complexity algorithm that successfully can use the existing correlation in order to determine a part of the secret key can be a very efficient way of attacking such stream ciphers in cryptanalysis. After the initializing ideas of Siegenthaler [12,13], Meier and Staffelbach [7,8] found a very interesting way of exploring the correlation in a fast correlation attack *provided that the feedback polynomial of the LFSR has a very low weight.* This work was followed by several papers, providing minor improvements to the initial results of Meier and Staffelbach, see [10,1,2,11]. For a recent application, see [14]. However, the algorithms that are efficient (good performance and low complexity) still require the feedback polynomial to be of low weight. Due to this requirement, it is today a general advise when constructing stream ciphers that the generator polynomial should not be of low weight.

The problem addressed in this paper is the problem of constructing algorithms achieving the similar performance and similar low complexity as mentioned above *but for any feedback polynomial.* The new algorithms that we propose are based on an interesting observation, namely that one can identify an embedded low-rate convolutional code in the code generated by the LFSR sequences. This embedded convolutional code can then be decoded with low complexity, using the Viterbi algorithm. From the result of the decoding phase, the secret key can be obtained. These algorithms provide a remarkable improvement over previous methods. As a particular example taken from [10], consider a LFSR of length 40 with a weight 17 feedback polynomial, and an observed sequence of length $4 \cdot 10^5$ bits. Let $1-p$ be the correlation probability. Then the algorithm in [7,8] and the improvement in [10] are successful up to $p \leq 0.104$ and $p \leq 0.122$, respectively, whereas the proposed algorithm is successful up to more than $p \leq 0.4$ with similar computational complexity.

The paper is organized as follows. In Section 2 we give some preliminaries on the decoding model that is used for cryptanalysis, and in Section 3 we shortly

review some previous algorithms for fast correlation attacks. In Section 4 we present our new ideas and give a description of the proposed algorithm. In Section 5 the simulation results are presented, and finally, in Section 6 we give some conclusions and possible extensions.

2 Preliminaries

Consider the model shown in Figure 2. As most other authors [13,7,8,10,1], we use the approach of viewing the problem as a decoding problem. Let the LFSR have length l and let the set of possible LFSR sequences be denoted by $\mathcal{L}$. Clearly, $|\mathcal{L}| = 2^l$ and for a fixed length N the truncated sequences from $\mathcal{L}$ is also a linear $[N, l]$ block code [6], referred to as $\mathcal{C}$. Furthermore, the keystream sequence $\mathbf{z} = z_1, z_2, \ldots, z_N$ is regarded as the received channel output and the LFSR sequence $\mathbf{u} = u_1, u_2, \ldots, u_N$ is regarded as a codeword from $\mathcal{C}$. Due to the correlation between u_i and z_i, we can describe each z_i as the output of the binary symmetric channel, BSC, when u_i was transmitted. The correlation probability $1 - p$, defined by $1 - p = P(u_i = z_i)$, gives p as the crossover probability (error probability) in the BSC. W.l.o.g we can assume $p < 0.5$. This is all shown in Figure 4.

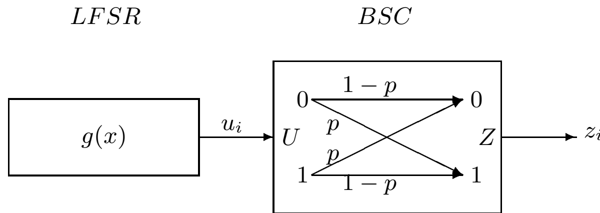

Fig. 4. Model for a correlation attack

The cryptanalyst's problem can be formulated as follows. Given a length N received word $(z_1, z_2, \ldots z_N)$ as output of the BSC(p), find the length N codeword from $\mathcal{C}$ that was transmitted.

From simple coding arguments, it can be shown that the length N should be at least around $N_0 = l/(1 - h(p))$ for unique decoding, where $h(p)$ is the binary entropy function. If the length of the output sequence N is modest but allows unique decoding, say $N = N_0 + D$, where D is a constant, the fastest methods for decoding are probabilistic decoding algorithms like Leon or Stern algorithms [5,15].

For received sequences of large length, $N \gg N_0$, *fast correlation attacks* [7,8] are sometimes applicable. These attacks resemble very much the iterative decoding process proposed by Gallager [3] for low-weight parity-check codes.

Due to the fact that the above attacks require the feedback polynomial $g(x)$ (or any multiple of $g(x)$ of modest degree) to have a low weight, one usually refrain from using such feedback polynomials in stream cipher design.

3 Fast Correlation Attacks – An Overview

In [7,8] Meier and Staffelbach presented two algorithms, referred to as A and B, for fast correlation attacks. Instead of an exhaustive search as originally suggested in [13], the algorithms are based on using certain parity check equations created from the feedback polynomial of the LFSR. All different algorithms for fast correlation attacks use two passes. In the first pass the algorithms find a set of suitable parity check equations in the code $\mathcal{C}$ stemming from the LFSR. The second pass uses these parity check equations in a fast decoding algorithm to recover the transmitted codeword and hence the initial state of the LFSR.

The set of parity check equations that was used in [7,8] was created in two separate steps. Let $g(x) = 1+g_1x^1+g_2x^2+\ldots+g_lx^l$ be the feedback polynomial, and t the number of taps of the LFSR, i.e., the weight of $g(x)$ (the number of nonzero coefficients) is $t+1$. Symbol number n of the LFSR sequence, u_n, can then be written as $u_n = g_1u_{n-1} + g_2u_{n-2} + \ldots + g_lu_{n-l}$. Since the weight of $g(x)$ is $t+1$, there are the same number of relations involving a fixed position u_n. Hence, we get in this way $t+1$ different parity check equations for u_n.

Secondly, using the fact that $g(x)^j = g(x^j)$ for $j = 2^i$, parity check equations are also generated by repeatingly squaring the polynomial $g(x)$. So if $g_0(x) = g(x)$, we create new polynomials by $g_{k+1}(x) = g_k(x)^2$, $k = 1, 2, \ldots$. This squaring is continued until the degree of a polynomial $g_k(x)$ is greater than the length N of the observed keystream. Each of the polynomials $g_k(x)$ are of weight $t+1$ and hence each gives $t+1$ new parity check equations for a fixed position u_n.

Combining this squaring technique with shifting the set of equations in time, the same parity check equations are essentially valid in each index position of $\mathbf{u}$. From [7,8] the number of parity check equations, denoted m, that can be found in this way is $m \approx \log(\frac{N}{2l})(t+1)$, where log uses base 2.

In the second pass, one writes the m equations for position u_n as,

$$\begin{aligned} u_n + b_1 &= 0, \\ u_n + b_2 &= 0, \\ &\vdots \\ u_n + b_m &= 0, \end{aligned} \tag{1}$$

where each b_i is the sum of t different positions of $\mathbf{u}$. Applying the same relations above to the keystream we can calculate the following sums,

$$\begin{aligned} z_n + y_1 &= L_1 \\ z_n + y_2 &= L_2 \\ &\vdots \\ z_n + y_m &= L_m. \end{aligned}$$

where y_i is the sum of the positions in the keystream corresponding to the positions in b_i. Assume that h out of the m equations in (1) hold, i.e.,

$$h = |\{i : L_i = 0, 1 \leq i \leq m\}|,$$

when we apply the equations to the keystream. Then it is possible to calculate the probability $p^* = P(u_n = z_n | h$ equation holds) as

$$p^* = \frac{ps^h(1-s)^{m-h}}{ps^h(1-s)^{m-h} + (1-p)(1-s)^h s^{m-h}},$$

where $p = P(z_n = a_n)$, and $s = P(b_i = y_i)$.

Using the parity check equations found above, two different decoding methods were suggested in [7,8]. The first algorithm, called Algorithm A, can shortly be described as follows: First find the equations to each position of the received bit and evaluate the equations. Then calculate the probabilities p^* for each bit in the keystream, select the l positions with highest value of p^*, and calculate a candidate initial state. Finally, find the correct value by checking the correlation between the sequence and the keystream for different small modifications of the candidate initial state.

The second algorithm, called Algorithm B, used another approach. Instead of calculating the probabilities p^* once and then make a hard decision, the probabilities are calculated iteratively. The algorithm uses two parameters p_{thr} and N_{thr}.

1. For all symbols in the keystream, calculate $p*$ and determine the number of positions N_w with $p^* < p_{thr}$.
2. If $N_w < N_{thr}$ repeat step 1 with p replaced by p^*.
3. Complement the bits with $p^* < p_{thr}$ and reset the probabilities to p.
4. If not all equations are satisfied go to step 1.

The performance of the algorithms described above is given in [7,8]. The algorithms above work well when the LFSR contains few taps, but for LFSRs with many taps the algorithms fail. The reason for this failure is that for LFSRs with many taps each parity check equation gives a very small average correction and hence many equations are needed. An improvement was suggested in [10], where a new method for finding parity check equations was suggested. Let $\mathbf{u}_0$ be the initial state of the LFSR. The state after t shifts can be written as $\mathbf{u}_t = A^t \mathbf{u}_0$, where A is an $l \times l$ matrix that depends of the feedback polynomial. Using powers of the matrix A a set of parity check equations can be found.

Another method of finding parity check equations was suggested in [1]. The idea of this algorithm is to use an algorithm for finding codewords of low weight in a general linear code.

4 New Fast Correlation Attacks Based on Convolutional Codes

The general idea behind the algorithm to be proposed can be described as follows. Looking at the parity check equations as described in (1), they are designed for a

second pass that consists of a very simple memoryless decoding algorithm. For a general feedback polynomial, this puts very hard restrictions on the parity check equations that can be used in (1) (weight $\leq t+1$ for a very low t). Our approach considers slightly more advanced decoding algorithms *that include memory*, but still have a low decoding complexity. This allows us to have looser restrictions on the parity check equations that can be used, leading to many more, and more, powerful equations. This work uses the Viterbi algorithm with memory $10-16$ as its decoding algorithm. The corresponding restrictions on the parity check equations will be apparent in the sequel.

The proposed algorithm transforms a part of the code $\mathcal{C}$ stemming from the LFSR sequences into a convolutional code. The encoder of this convolutional code is created by finding suitable parity check equations from $\mathcal{C}$. Some notation and basic concepts regarding convolutional codes that are frequently used can be found in Appendix A.

The convolutional code will have rate $R = 1/(m+1)$, where the constant $(m+1)$ will be determined later. Furthermore, let B be a fixed memory size. In a convolutional encoder with memory B the vector $\mathbf{v}_n$ of codeword symbols at time n is of the form

$$\mathbf{v}_n = u_n G_0 + u_{n-1} G_1 + \ldots u_{n-B} G_B, \tag{2}$$

where in the case $R = 1/(m+1)$ each G_i is a vector of length $(m+1)$. The task in the first pass of the algorithm is to find suitable parity check equations that will determine the vectors $G_i, 0 \leq i \leq m$, defining the convolutional code.

Let us start with the linear code $\mathcal{C}$ stemming from the LFSR sequences. There is a corresponding $l \times N$ generator matrix G_{LFSR}. Clearly, $\mathbf{u} = \mathbf{u}_0 G_{LFSR}$, where $\mathbf{u}_0$ is the initial state of the LFSR. The generator matrix is furthermore written on systematic form, i.e., $G_{LFSR} = \begin{pmatrix} I_l & Z \end{pmatrix}$, where I_l is the $l \times l$ identity matrix. Given a generator matrix on this form, the parity check matrix is written as $P_{LFSR} = \begin{pmatrix} Z^T & I_{N-l} \end{pmatrix}$, where each row of P defines a parity check equation in $\mathcal{C}$.

We are now interested in finding parity check equations that involve a current symbol u_n, an arbitrary linear combination of the B previous symbols $u_{n-1}, \ldots, u_{n-B}$, together with at most t other symbols. Clearly, t should be small and we mainly consider $t = 2$.

To find these equations we start by considering the index position $n = B+1$. Introduce the following notation for the generator matrix,

$$G_{LFSR} = \begin{pmatrix} I_{B+1} & Z_{B+1} \\ 0_{l-B-1} & Z_{l-B-1} \end{pmatrix}. \tag{3}$$

Parity check equations for u_{B+1} with weight t outside the first $B+1$ positions can then be found by finding linear combinations of t columns of Z_{l-B-1} that add to the all zero column vector. This corresponds to the problem of finding weight t codewords in the code dual to Z_{l-B-1}.

For the case $t = 2$ the parity check equations can be found in a very simple way as follows. A parity check equation with $t = 2$ is found if two columns from

G_{LFSR} have the same value when restricted to the last $l-B-1$ entries (the Z_{l-B-1} part). Hence, we simply put each column of Z_{l-B-1} into one of 2^{l-B-1} different "buckets", sorted according to the value of the last $l-B-1$ entries. Each pair of columns in each bucket will provide us with one parity check equation, provided u_{B+1} is included.

Assume that the above procedure gives us a set of m parity check equations for u_{B+1}, written as

$$\begin{array}{c} u_{B+1} + \sum_{i=1}^{B} c_{i1} u_{B+1-i} + \sum_{i=1}^{\leq t} u_{j_{i1}} = 0, \\ u_{B+1} + \sum_{i=1}^{B} c_{i2} u_{B+1-i} + \sum_{i=1}^{\leq t} u_{j_{i2}} = 0, \\ \vdots \\ u_{B+1} + \sum_{i=1}^{B} c_{im} u_{B+1-i} + \sum_{i=1}^{\leq t} u_{j_{im}} = 0. \end{array}$$

Now it follows directly from the cyclic structure of the LFSR sequences that *exactly* the same set of parity checks is valid for any index position n simply by shifting all the symbols in time, resulting in

$$\begin{array}{c} u_n + \sum_{i=1}^{B} c_{i1} u_{n-i} + b_1 = 0, \\ u_n + \sum_{i=1}^{B} c_{i2} u_{n-i} + b_2 = 0, \\ \vdots \\ u_n + \sum_{i=1}^{B} c_{im} u_{n-i} + b_m = 0, \end{array} \tag{4}$$

where $b_k = \sum_{i=1}^{\leq t} u_{j_{ik}}$, $1 \leq k \leq m$ is the sum of (at most) t positions in $\mathbf{u}$.

Using the equations above we next create an $R = 1/(m+1)$ bi-infinite systematic convolutional encoder. Recall that the generator matrix for such a code is of the form

$$\mathbf{G} = \begin{pmatrix} \ddots & \ddots & & \ddots & & \\ & G_0 & G_1 & \dots & G_B & \\ & & G_0 & G_1 & \dots & G_B \\ & & & \ddots & \ddots & & \ddots \end{pmatrix}, \tag{5}$$

where the blank parts are regarded as zeros. Identifying the parity check equations from (4) with the description form of the convolutional code as in (5) gives us

$$\begin{pmatrix} G_0 \\ G_1 \\ \vdots \\ G_B \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 0 & c_{11} & c_{12} & \dots & c_{1m} \\ 0 & c_{21} & c_{22} & \dots & c_{2m} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & c_{B1} & c_{B2} & \dots & c_{Bm} \end{pmatrix}. \tag{6}$$

For each defined codeword symbol $v_n^{(i)}$ in the convolutional code we have an estimate of that symbol from the transmitted sequence $\mathbf{z}$.

Consider $t = 2$. If $v_n^{(i)} = u_n$ (an information bit) then $P(v_n^{(i)} = z_n) = 1 - p$. Otherwise, if $v_n^{(i)} = u_{j_{1i}} + u_{j_{2i}}$ from (4) then $P(v_n^{(i)} = z_{j_{1i}} + z_{j_{2i}}) = (1-p)^2 + p^2$. Using these estimates we can construct a sequence

$$\mathbf{r} = \ldots r_n^{(0)} r_n^{(1)} \ldots r_n^{(m)} r_{n+1}^{(0)} r_{n+1}^{(1)} \ldots r_{n+1}^{(m)} \ldots ,$$

where $r_n^{(0)} = z_n$ and $r_n^{(i)} = z_{j_{1i}} + z_{j_{2i}}$, $1 \le i \le m$, that plays the role of a received sequence for the convolutional code. Then we have from the estimates that $P(v_n^{(0)} = r_n^{(0)}) = 1 - p$ and that $P(v_n^{(i)} = r_n^{(i)}) = (1-p)^2 + p^2$ for $1 \le i \le m$. Next, we enter the decoding phase.

To recover the initial state of the LFSR it is enough to decode l consecutive information bits correctly. Optimal decoding (ML decoding) of convolutional codes uses the Viterbi algorithm to decode.

The original Viterbi algorithm assumes that the convolutional encoder starts in state $\mathbf{0}$. However, in our application there is neither a starting state, nor an ending state. To deal with this we start by assigning the metrics $\log P(\mathbf{s} = z_1, z_2, \ldots, z_B)$ to each state $\mathbf{s}$ in the trellis. We then proceed to decode from $n = B$ as usual. Due to the difference regarding the endpoints, we run the Viterbi algorithm over a number of "dummy" information symbols, before we come to the l information symbols that we try to decode correctly. Similarly, after these l information symbols we continue the Viterbi algorithm over another set of "dummy" information symbols before the algorithm outputs the result. These are well known techniques in Viterbi decoding, and typically one has to decode approximately $4-5$ times B "dummy" information symbols, [4], before making the decision. This means that decoding takes place over approximately $J = l + 10B$ information symbols, where the l symbols in the middle are regarded as the l bit sequence that we want to estimate. This estimate from the Viterbi algorithm is then used to provide the corresponding estimate of the initial state of the LFSR. This conclude the general description and we give a detailed summary of the algorithm for $t = 2$.

The Proposed Algorithm ($t = 2$)

Input: The systematic $l \times N$ generator matrix in the form

$$G_{LFSR} = \left(I_{B+1}\ \mathbf{g}_{B+2} \cdots \mathbf{g}_J\ \mathbf{g}_{J+1} \cdots \mathbf{g}_N \right).$$

1. For $J + 1 \le i, j \le N$ find all pairs of columns $\mathbf{g}_i, \mathbf{g}_j$ such that

$$(\mathbf{g}_i + \mathbf{g}_j)^T = (\underbrace{*, *, \ldots, *}_{B}, 1, \underbrace{0, 0, \ldots, 0}_{l-B-1}),$$

where $*$ means an arbitrary value. Then add

$$(u_{n-B}, u_{n-B-1}, \ldots, u_n, 0, 0, \ldots, 0) \cdot (\mathbf{g}_i + \mathbf{g}_j) + u_{n+i} + u_{n+j} = 0$$

to the set of parity check equations as in (4).

2. From this set, calculate $G_0, G_1, \ldots, G_B$ as in (6).
   Create a received vector $\mathbf{r}$ from $\mathbf{z}$ by $r_n^{(0)} = z_n$ and $r_n^{(i)} = z_{j_{1i}} + z_{j_{2i}}$ for $1 \leq i \leq m$, where j_{1i} and j_{2i} are the indices determined in 1.
3. Let $P(v_n^{(0)} = r_n^{(0)}) = 1 - p$ and $P(v_n^{(i)} = r_n^{(i)}) = (1-p)^2 + p^2$ for $B + 1 \leq n \leq l + 10B$.
   Decoding part
4. For each state $\mathbf{s}$, let $\log(P(\mathbf{s} = (z_1, z_2 \cdots, z_B))$ be the initial metric for that state when we start the Viterbi algorithm at $n = B$.
5. Decode the received sequence $\mathbf{r}$ using the Viterbi algorithm from $n = B$ until $n = J$. Output the estimated information sequence $(\hat{u}_{5B+1}, \hat{u}_{5B+2}, \ldots, \hat{u}_{5B+l})$. Finally, calculate the corresponding initial state of the LFSR.

An Illustrating Example

Consider a length 40 LFSR, with feedback polynomial

$$g(x) = 1 + x + x^3 + x^5 + x^9 + x^{11} + x^{12} + x^{17} + x^{19} + x^{21} + x^{25} + x^{27} + x^{29} + x^{32} + x^{33} + x^{38} + x^{40}.$$

An observed key sequence $\mathbf{z}$ of length $N = 40000$ is found to be correlated to the LFSR sequence with probability $1 - p = 1 - 0.1$. We want to decode the received sequence $\mathbf{z}$ transmitted over BSC(0.1) using the proposed method with memory $B = 10$.

We start by writing down the generator matrix G_{LFSR}. Then we search for suitable parity check equations by finding all pairs of columns in G_{LFSR} for which the last 29 index positions are all zero. Each such pair gives rise to one parity check equation with $t = 2$. In this case, the following three parity check equations were found

$$u_n + u_{n-1} + u_{n-8} + u_{n-10} + u_{n+4690} + u_{n+23655} = 0,$$
$$u_n + u_{n-2} + u_{n-3} + u_{n-4} + u_{n-7} + u_{n-8} + u_{n+4817} + u_{n+31970} = 0,$$
$$u_n + u_{n-2} + u_{n-3} + u_{n-4} + u_{n-5} + u_{n-9} + u_{n+18080} + u_{n+4626} = 0,$$

which are all valid for $1 \leq n \leq 8030$. We get a fourth codeword symbol by the information symbol u_n itself. Then we can identify

$$\begin{pmatrix} G_0 \\ G_1 \\ \vdots \\ G_B \end{pmatrix} = \begin{pmatrix} 1111 \\ 0100 \\ 0011 \\ \vdots \\ 0100 \end{pmatrix}.$$

Thus, we have created a rate $R = 1/4$ convolutional code having generator matrix

$$\mathbf{G} = \begin{pmatrix} \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & 1111 & 0100 & 0011 & 0011 & 0011 & 0001 & 0000 & 0010 & 0110 & 0001 & 0100 & \\ & & 1111 & 0100 & 0011 & 0011 & 0011 & 0001 & 0000 & 0010 & 0110 & 0001 & 0100 \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{pmatrix}.$$

Each $\mathbf{r}_n$ in the received sequence $\mathbf{r} = \mathbf{r}_0\mathbf{r}_1\dots$ for the convolutional code is created as

$$\begin{aligned} r_n^{(0)} &= z_n, \\ r_n^{(1)} &= z_{n+4690} + z_{n+23655}, \\ r_n^{(2)} &= z_{n+4817} + z_{n+31970}, \\ r_n^{(3)} &= z_{n+4626} + z_{n+18080}, \end{aligned}$$

and $P(v_n^{(0)} = r_n^{(0)}) = 0.9$ and $P(v_n^{(i)} = r_n^{(i)}) = 0.82, 1 \leq i \leq 3$. Finally we run the Viterbi algorithm, starting in $n = 10$ with all 2^{10} different states $(u_1, u_2, \dots, u_{10})$. Each state have the initial metric $\log(P(u_1 = z_1)P(u_2 = z_2)\cdots P(u_B = z_B))$. After reaching $n = 140$, we output $(\hat{u}_{51}, \hat{u}_{52}, \dots, \hat{u}_{90})$.

5 Simulation Results

In this section we present some simulation results for our algorithm. The obtained results are compared with the received results in [7,8,10]. We choose to use exactly the same case as tabulated in [10]. Thus all the simulations are based on a LFSR with length $l = 40$, and a weight 17 feedback polynomial which is

$$\begin{aligned} g(x) = 1 + x + x^3 + x^5 + x^9 + x^{11} + x^{12} + x^{17} + x^{19} + \\ x^{21} + x^{25} + x^{27} + x^{29} + x^{32} + x^{33} + x^{38} + x^{40}. \end{aligned}$$

	[7,8]	[10]	Our Algorithm		
N/l	Alg. B	Alg.	$B = 13$	$B = 14$	$B = 15$
10^3	0.092	0.096	0.19	0.22	0.26
10^4	0.104	0.122	0.37	0.39	0.40

Table 1. Maximum p for different algorithms.

In Table 1 the maximum crossover probability p is shown for algorithm B in [7,8], the improvement in [10], and the proposed algorithm. Our results are generated for different sizes of the memory B. As a particular example, we can see that when we have $4 \cdot 10^5$ received symbols the proposed algorithm is successful up to more than $p = 0.4$ for memory $B = 15$, whereas the algorithm in [7,8] and the improvement in [10] are successful only up to a crossover probability of 0.104 and 0.122, respectively. In this case, $B = 15$, the proposed algorithm finds roughly 2300 parity checks and hence the embedded convolutional code is of rate roughly $R = 1/2300$. Also, the decoding takes place over $J = 200$ information symbols. The computational complexity is proportional to $J \cdot m \cdot 2^B$, and in the case $B = 15, M = 2300, J = 200$ the whole attack takes less than one hour on a PC.

Another interesting property to look at is the success rate, i.e., the probability for successful decoding given a channel with crossover probability p. In Figure 5 we plot the success rate as a function of p, when $B = 14$, for 40000 and 400000 received symbols, respectively.

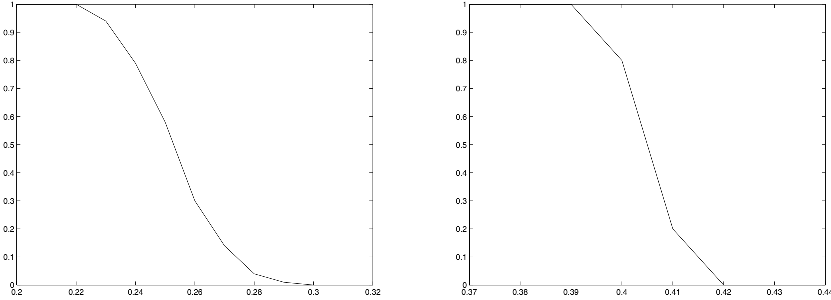

Fig. 5. Success rate for $B = 14$ with $N = 40000$ and $N = 400000$.

Finally, we make a comment regarding the theoretical performance of the proposed algorithm for $t = 2$. For fixed parameters l, B and N, we can determine the expected number of suitable parity checks, i.e., the parameter m. Then one can show that the success rate will be very close to 1 if the rate $R = 1/(m+1)$ is below the *cutoff rate* R_0 [4] for the BSC$(2p(1-p))$. However, we observe that the simulated results are very close to the capacity C of the BSC$(2p(1-p))$, which is $C = 1 - h(2p(1-p))$.

6 Conclusions

New methods for fast correlation attacks have been proposed, based on identifying an embedded convolutional code in the code $\mathcal{C}$ generated by the LFSR sequences of a fixed length N. The results show a significant improvement compared with previous work regarding general feedback polynomials. We have described the methods using an ordinary convolutional code together with standard Viterbi decoding. There are many different ways to extend these methods that can be considered in future work.

Firstly, we note that by permuting the columns of $\mathcal{C}$ before searching for parity checks, we receive a *time-varying* convolutional code. Secondly, the computational complexity of the Viterbi algorithm is growing exponentially with B, which means that in practice B is bounded to be at most $20 - 30$. But there are several other decoding algorithms, which are not ML, that have a much lower computational complexity. Examples of such algorithms are the M-algorithm (list decoding) and different sequential decoding algorithms [4]. They are promising candidates for improving the performance.

Finally, we also mention the possibility of using iterative decoding. This can roughly be described as follows. Identify several convolutional codes in $\mathcal{C}$ that have certain codeword symbols in common. Then decode them using APP (a posteriori probability) decoding algorithms [4] and pass the symbol probabilities to the other decoders. This procedure is iterated until the symbol probabilities have converged to 0 or 1. We believe that this is a very promising approach, and that we might see a further improvement in performance compared to the results in this paper.

References

1. 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.
2. 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.
3. R. G. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge, MA, 1963.
4. R. Johannesson, K. Sh. Zigangirov, *Fundamentals of Convolutional Codes,* IEEE Press, New York, 1999.
5. J. Leon, "A probabilistic algorithm for computing minimum weights of large error-correcting codes", *IEEE Trans. Information Theory*, vol. IT–34, 1988, pp. 1354–1359.
6. F. MacWilliams, N. Sloane, *The Theory of Error Correcting Codes*, North Holland, 1977.
7. 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.
8. W. Meier, and O. Staffelbach, "Fast correlation attacks on certain stream ciphers", *Journal of Cryptology*, vol. 1, 1989, pp. 159–176.
9. A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
10. 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.
11. 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.
12. T. Siegenthaler, "Correlation-immunity of nonlinear combining functions for cryptographic applications", *IEEE Trans. on Information Theory*, vol. IT–30, 1984, pp. 776–780.
13. T. Siegenthaler, "Decrypting a class of stream ciphers using ciphertext only", *IEEE Trans. on Computers,* vol. C–34, 1985, pp. 81–85.

14. L. Simpson, E. Dawson, J. Golic, M. Salamasizadeh, "Fast correlation attacks on the multiplexer generator", *Proc. IEEE 1998 International Symposium on Information Theory, ISIT'98*, 1998, p. 270.
15. J. Stern, "A method for finding codewords of small weight," *Coding Theory and Applications*, Springer-Verlag, 1989, pp. 106–113.

A Convolutional Codes

This section reviews some basic concepts regarding convolutional codes. For a more thorough treatment we refer to [4]. A convolutional code is a linear code where the information symbols and the codeword symbols are treated as infinite sequences. In a general rate $R = b/c$, $b \leq c$ binary convolutional encoder (time-invariant and without feedback) the causal information sequence

$$\mathbf{u} = \mathbf{u_0 u_1} \ldots = u_0^{(0)} u_0^{(1)} \ldots u_0^{(b)} u_1^{(0)} u_1^{(1)} \ldots u_1^{(b)} \ldots$$

is encoded as the causal code sequence

$$\mathbf{v} = \mathbf{v_0 v_1} \ldots = v_0^{(0)} v_0^{(1)} \ldots v_0^{(c)} v_1^{(0)} v_1^{(1)} \ldots v_1^{(c)} \ldots ,$$

where

$$\mathbf{v_t} = f(\mathbf{u_t}, \mathbf{u_{t-1}}, \ldots , \mathbf{u_{t-B}}).$$

The function f must be a linear function. Furthermore, the parameter B is called the encoder memory.

In our particular application we only consider convolutional codes for which the rate is of the form $R = 1/c$, i.e., $b = 1$, and thus we now adopt the notation

$$\mathbf{u} = u_0 u_1 \ldots ,$$

where $u_i \in \mathbb{F}_2$. Since f is a linear function, it is convenient to write

$$\mathbf{v_t} = u_t G_0 + u_{t-1} G_1 + \cdots + u_{t-B} G_B,$$

where G_i, $0 \leq i \leq B$ is a $1 \times c$ matrix, i.e., a length c vector. Now we can rewrite the expression for the code sequence as

$$\mathbf{v_0 v_1} \ldots = (u_0 u_1 \ldots)\mathbf{G},$$

where

$$\mathbf{G} = \begin{pmatrix} G_0 & G_1 & \ldots G_B & & \\ & G_0 & G_1 & \ldots G_B & \\ & & \ddots & \ddots & \ddots \end{pmatrix}, \tag{7}$$

and the blank parts of $\mathbf{G}$ is assumed to be filled with zeros. We call $\mathbf{G}$ the *generator matrix.* The encoder can be illustrated as in Figure 6.

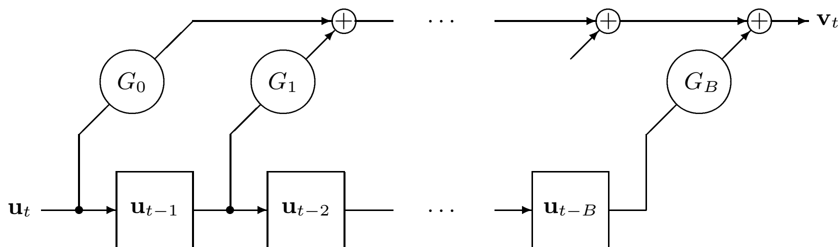

Fig. 6. A general convolutional encoder (without feedback).

The *state* of a system is a description that together with a specification of the present and future inputs, can determine the present and future outputs. From Figure 6 it is easy to see that we can choose the contents of the memory cells at time t as the encoder state $\sigma_{\mathbf{t}}$ at time t,

$$\sigma_{\mathbf{t}} = u_{t-1}u_{t-2}\dots u_{t-B}.$$

Thus the encoder has at most 2^B different states at each time instant. We can now consider all possible states $\sigma_{\mathbf{t}}$ as vertices in a graph and put an edge between two adjacent states $\sigma_{\mathbf{t}}$ and $\sigma_{\mathbf{t+1}}$ if and only if there is an information symbol u_t such that takes the state from $\sigma_{\mathbf{t}}$ at time t to $\sigma_{\mathbf{t+1}}$ at time $t+1$. This graph gives rise to a so called *trellis*. The *convolutional code* (or linear trellis code) is the set of all possible codeword sequences (possibly with a predetermined starting and ending state). If we label the edge in the trellis going from $\sigma_{\mathbf{t}}$ to $\sigma_{\mathbf{t+1}}$ with $\mathbf{v_t} = u_tG_0 + u_{t-1}G_1 + \dots + u_{t-B}G_B$ the set of codeword sequences will correspond to the set of possible paths in the trellis.

Example: Consider the rate $R = 1/2$ convolutional encoder with generator matrix

$$\mathbf{G} = \begin{pmatrix} 11 & 10 & 11 & & \\ & 11 & 10 & 11 & \\ & & \ddots & \ddots & \ddots \end{pmatrix}.$$

The encoder can be implemented as in Figure 7, and the corresponding trellis is depicted in Figure 8.

Suppose now that our trellis code is transmitted over the BSC with error probability p. We are interested in determining the most probable codeword from a received sequence $\mathbf{r}$,

$$\mathbf{r} = \mathbf{r_0}\mathbf{r_1}\dots = r_0^{(0)}r_0^{(1)}\dots r_0^{(c)}r_1^{(0)}r_1^{(1)}\dots r_1^{(c)}\dots.$$

This corresponds to a maximum likelihood decoding problem, ML decoding. The solution to the ML decoding problem for convolutional codes is the famous *Viterbi algorithm*.

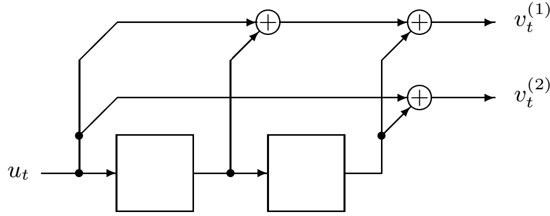

Fig. 7. A rate $R = 1/2$ convolutional encoder.

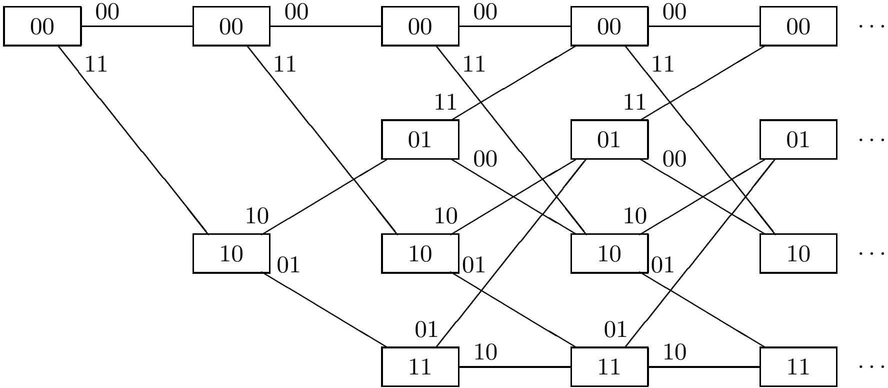

Fig. 8. A binary rate $R = 1/2$ trellis code.

The ML decoder chooses as its estimate $\hat{\mathbf{v}}$ a sequence $\mathbf{v}$ that maximizes $P(\mathbf{r}|\mathbf{v})$. Assuming that the starting and ending state is predetermined to be the zero-state, the ML decoder works as follows. Introduce the Viterbi branch metric, $\mu(\mathbf{r_n}, \mathbf{v_n}) = \sum_i \log P(r_n^{(i)}|v_n^{(i)})$ (One usually introduce a translation and a scaling in order to approximate the metric values with suitable integers [4]).

The Viterbi Algorithm

1. Assign the Viterbi metric to be zero at the initial node, and set $n = 0$.
2. For each node at depth $n+1$: Find for each of its predecessors at depth n the sum of the metric of the predecessor and the branch metric of the connecting branch. Find the maximum sum and assign this metric value to the node. Also, label the node with the shortest path to it.
3. If we have reached the end of the trellis, stop and choose as the estimate $\hat{\mathbf{v}}$ a path to the ending node with largest Viterbi metric; otherwise increment n and go to 2.