Improved Fast Correlation Attack Using Low Rate Codes

Håvard Molland, John Erik Mathiassen, and Tor Helleseth

The Selmer Center* Dept. of Informatics, University of Bergen
P.B. 7800 N-5020 BERGEN, Norway
{molland,johnm,tor.helleseth}@ii.uib.no

Abstract. In this paper we present a new and improved correlation attack based on maximum likelihood (ML) decoding. Previously the code rate used for decoding has typically been around $r = 1/2^{14}$. Our algorithm has low computational complexity and is able to use code rates around $r = 1/2^{33}$. This way we get much more information about the key bits. Furthermore, the run time for a successful attack is reduced significantly and we need fewer key stream bits.

1 Introduction

Linear feedback shift registers, $LFSRs$, are popular building blocks for stream ciphers, since they are easy to implement, easy to analyze, and they have nice cryptographic properties. But a linear shift register is not a cryptographic secure function in itself. Assuming we know the connection points in the LFSRs, we just need to know a few bits of the key stream to find the key bits, by using the linear properties in the streams to set up an equation set that is easily solved.

To make such a cipher system more secure, it is possible to combine n LFSRs with a nonlinear function f in such a way that linear complexity becomes very high. Fig 1 describes an example for this model. The key stream $\mathbf{z} = (z_0, z_1, ..., z_t, ..., z_{N-1})$ is generated by $z_t = f(u_t^1, u_t^2, ..., u_t^n)$ and the linearity in the bit streams $\mathbf{u}^i = (u_0^i, u_1^i, ..., u_t^i, ..., u_{N-1}^i)$ from the n LFSRs is destroyed. The plain text $\mathbf{m}$ of length N is then encrypted to cipher text $\mathbf{c}$ by $c_t = z_t \oplus m_t$, $0 \leq t < N$.

There exist different types of attacks on systems based on this scheme. The type of attack we describe in this paper is the correlation attack. The attack uses the fact that there often exist some correlations between the bits in some of the shift register streams and the key stream $\mathbf{z}$. This can be formulated as the crossover probability $p = P(u_t \neq z_t)$, where u_t is the bit stream from a LFSR that has a correlation with $\mathbf{z}$. When $p \neq 0.5$, it is possible to do a correlation attack. If $p = 0.5$ there would be no correlation, and a correlation attack could not be done. But it is a well known fact that there always exists a correlation between sums of $\mathbf{u}$ and $\mathbf{z}$ in the model described in Fig. 1. That is

* This work was supported by the Norwegian Research Council under Grant 146874/420.

K.G. Paterson (Ed.): Cryptography and Coding 2003, LNCS 2898, pp. 67–81, 2003.

$P(u_{t+j_1} + u_{t+j_2} + ... + u_{t+j_M} \neq z_t) \neq 0.5$ for given M and $(j_1, j_2, ..., j_M)$. When we add a LFSR bit stream with a shift of itself, we always get a new LFSR bit stream. Thus, the model is to decode a LFSR stream that has been sent through a binary symmetric channel (BSC) with crossover probability p.

The simplest correlation attack[6] chooses the shift register LFSR_i that has a correlation to the key stream bit z. Then the initialization bits $\hat{\mathbf{u}}^I$ for the LFSR are guessed and the bit stream $\hat{\mathbf{u}} = (\hat{u}_0, \hat{u}_1, ..., \hat{u}_{N-1})$ is generated. If for a chosen threshold p_{tr} there exists a correlation between the guessed bit stream $\hat{\mathbf{u}}$ and $\mathbf{z}$ such that $P(u_t \neq z_t) < p_{tr} < 0.5$ for $0 \leq t < N$, it is assumed that the correct initialization bits are found. This attack has a complexity of $O(2^{l_i} \cdot N)$ which is much better than $O(2^{l_1+l_2+...+l_n})$, the complexity for guessing the initialization bits for all the LFSRs.

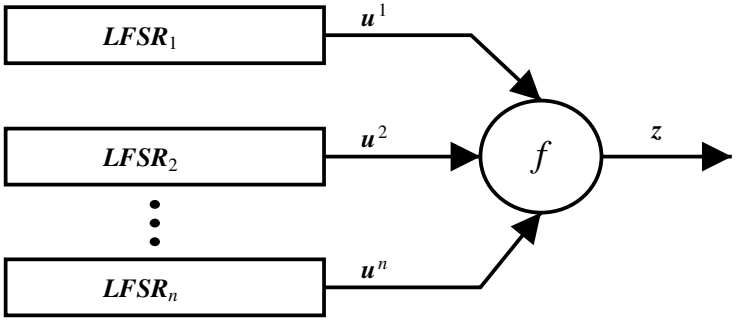

Fig. 1. An example of a stream cipher we are are able to attack using fast correlation attacks. The linear feedback shift registers LFSR_i of length l_i, for $1 \leq i \leq N$, are sent through a nonlinear function f to generate the key stream $\mathbf{z}$

The complexity for guessing all the bits in a given LFSR_i can be too high. To get around this, the fast correlation attack was developed[7,8] by Meier and Staffelbach. This attack uses parity check equations and reconstructs $\mathbf{u}$ from $\mathbf{z}$ using an iterative decoding algorithm. The attack works well when the polynomial that defines the LFSR_i has few taps, but fails when the polynomial has many taps.

In [4] Johansson and Jönsson presented a better attack that works for LFSRs with many taps. Using a clever search algorithm, they find parity equations that are suitable for convolutional codes. The decoding is done using the Viterbi algorithm, which is maximum likelihood. This attack is briefly explained in Sect. 2.

In [1] David Wagner found a new algorithm to solve the generalized birthday problem. In this paper we present an algorithm based on the same idea that finds many equations suitable for correlation attacks. The problem with this algorithm is that it finds many but weak equations, and previous attacks would not be very effective since the code rate will be very low.

In this paper we present an improvement on the attacks based on ML decoding. While Johansson and Jönsson use few but strong equations, we go in the opposite direction and use many and weak equations. We present a new algorithm that is capable of performing an efficient ML decoding even when the code rate is very low. This gives us much more information about the secret initialization bits, and the run time complexity goes down considerably. For a crossover probability $p = 0.47$, polynomial of degree $l = 60$ and the number of known key stream bits $N = 100 \cdot 10^6$, our attack has complexity of order 2^{39}, while the previous convolutional code attack[4,5] has complexity of order 2^{48}. See Table 2 in Sect. 5 for more simulation results compared to previous attacks.

The paper will be organized as follows. First we will give a brief description of the systems we try to attack. In Sect. 2 we will describe the basic mathematics and some important previous attacks. In Sect. 3 we describe an efficient method for finding parity check equations, using the generalized birthday problem. In Sect. 4 we present a new algorithm that is capable of using the huge number of equations found by the method in Sect. 3.

2 Definitions and Previous Attacks

First we will define the basic mathematics for the correlation attacks in this paper.

2.1 The Generator Matrix

Let $g(x) = 1 + g_{l-1}x + g_{l-2}x^2 + ... + g_1x^{l-1} + x^l$ be the primitive feedback polynomial over $\mathbb{F}_2$ of degree l for a linear feedback register, LFSR, that generates the sequence $u = (u_0, u_1, ..., u_{N-1})$. The corresponding recurrence is $u_t = g_1u_{t-1} + g_2u_{t-2} + u_{t-l}$. Let α be defined by $g(\alpha) = 0$. From this we get the reduction rule $\alpha^l = g_1\alpha^{l-1} + g_2\alpha^{l-2} + ... + g_{l-1}\alpha + 1$. Then we can define the generator matrix for sequence u_t, $0 < t < N$ by the $l \times N$ matrix

$$G = [\alpha^0\alpha^1\alpha^2...\alpha^{N-1}]. \tag{1}$$

For each $i > l$, using the reduction rule, α^i can be written as $\alpha^i = h^i_{l-1}\alpha^{l-1} + ... + h^i_2\alpha^2 + h^i_1\alpha + h^i_0$. We see that every column $i \geq l$ is a combination of the first l columns. Any column i in G can be represented by

$$\mathbf{g}_i = [h^i_0, h^i_1, ..., h^i_{l-1}]^{\mathbf{T}}. \tag{2}$$

Thus the sequence u with length N and initialization bits $\mathbf{u}^I = (u_0, u_1, ..., u_{l-1})$, can be generated by

$$\mathbf{u} = \mathbf{u}^I G.$$

The shift register is now turned into a (N, l) block code.

Example 1. Let $g(x) = x^4 + x^3 + 1$. Using the reduction rule we get $\alpha^4 = \alpha^3 + 1$, $\alpha^5 = \alpha(\alpha^3 + 1) = \alpha^4 + \alpha = \alpha^3 + \alpha + 1$ and so on. We choose $N = 10$, and set $G = [\alpha^0 \alpha^1 ... \alpha^9]$. The sequence $\mathbf{u}$ is generated by the 4×10 matrix G like this,

$$\mathbf{u} = \mathbf{u}^I G = [u_0, u_1, u_2, u_3] \begin{bmatrix} 1\,0\,0\,0\,1\,1\,1\,1\,0\,1 \\ 0\,1\,0\,0\,0\,1\,1\,1\,1\,0 \\ 0\,0\,1\,0\,0\,0\,1\,1\,1\,1 \\ 0\,0\,0\,1\,1\,1\,1\,0\,1\,0 \end{bmatrix}. \tag{3}$$

The reason that we use a generator matrix, is that we easily can see from G which initialization bits $(u_0, u_1, ..., u_{l-1})$ sum to u_i for every $0 \leq i < N$ by looking at column i. For example the bit u_9 (last column) in the example above is calculated by $u_9 = u_0 + u_2$, and it is independent of the initialization bits u_1 and u_3.

2.2 Equations

In [4] Johansson and Jönsson presented the following method for finding equations that are usable for decoding.

Let u be a sequence generated by the generator polynomial $g(x)$ with degree l. If we can find w columns in the generator matrix G that summarize to zero in the $l - B$ last bits,

$$(\mathbf{g}_{i_1} + \mathbf{g}_{i_2} + \ldots + \mathbf{g}_{i_w})^{\mathbf{T}} = (c_0, c_1, ..., c_{B-1}, \underbrace{0, 0, ..., 0}_{l-B}), \tag{4}$$

for a given B, $0 < B \leq l$, and $l \leq i_1, i_2, ..., i_w < N$, we get an equation of the form

$$c_0 u_0 + c_1 u_1 + ... + c_{B-1} u_{B-1} = u_{i_1} + u_{i_2} + ... + u_{i_w}. \tag{5}$$

This can be seen by noting that column i in G shows which of the initialization bits $\mathbf{u}^I = (u_0, u_1, ..., u_{l-1})$ that summarize to the bit u_i in the sequence $\mathbf{u}$. When two columns i and j in G sum to zero in the last $l-B$ entries $(u_B, u_{B+1}, ..., u_{l-1})$, the sum $u_i + u_j$ is independent of those bits. Then we can concentrate on finding just the B first bit of $\mathbf{u}^I$. The equation (5) is cyclic and can therefore be written as

$$c_0 u_t + c_1 u_{t+1} + ... + c_{B-1} u_{t+B-1} = u_{t+i_1} + u_{t+i_2} + ... + u_{t+i_w}, \tag{6}$$

for $0 \leq t < N - i_w$.

Example 2. Let $w = 2$, and $B = 1$. If we examine the matrix G in equation (3), we see that $(\mathbf{g}_6 + \mathbf{g}_8)^{\mathbf{T}} = (1, 1, 1, 1) + (0, 1, 1, 1) = (1, 0, 0, 0)$. From this we get $c_0 = 1$, $i_1 = 6$ and $i_2 = 8$ and the equation is $u_0 = u_6 + u_8$. Because of the cyclic structure we finally get $u_t = u_{t+6} + u_{t+8}$. This equation will hold for every sequence that is generated with $g(x) = x^4 + x^3 + 1$ as feedback polynomial.

In section 3 we will go further into how the actual search for columns that sum to zero in the last $l - B$ bits can be done efficiently.

2.3 Principle for Decoding

In [2] Chepyzhov, Johansson and Smeets presented a simple maximum likelihood algorithm that uses equations found in Sect. 2.2 for decoding. We will now briefly describe this algorithm. First we take equation (5) and make the right side of the equation point to the corresponding key stream bits $\mathbf{z}$ instead of $\mathbf{u}$. From this we get the following equation,

$$c_0u_0 + c_1u_1 + ... + c_{B-1}u_{B-1} \approx z_{i_1} + z_{i_2} + ... + z_{i_w}. \tag{7}$$

Let m be the number of equations found by the method in Sect. 2.2. Then we get the equation set

$$\begin{aligned} c_{0,0}u_0 + c_{0,1}u_1 + \ldots + c_{0,B-1}u_{B-1} &\approx z_{i_{1,1}} + z_{i_{1,2}} + ... + z_{i_{1,w}} \\ c_{1,0}u_0 + c_{1,1}u_1 + \ldots + c_{2,B-1}u_{B-1} &\approx z_{i_{2,1}} + z_{i_{2,2}} + ... + z_{i_{2,w}}. \\ &\vdots \\ c_{m,0}u_0 + c_{m,1}u_1 + \ldots + c_{m,B-1}u_{B-1} &\approx z_{i_{m,1}} + z_{i_{m,2}} + ... + z_{i_{m,w}} \end{aligned} \tag{8}$$

We use $'\approx'$ to notify that the equations only hold with a certain probability.

Here $(u_0, u_1, ..., u_{B-1})$ are the unknown secret bits we want to find and $\mathbf{z}$ is the key stream. Remember that u_t and z_t are equal with a probability $1 - p$ where $p = P(u_t \neq z_t)$. Thus, each equation in (8) will hold with a probability

$$P_w = \frac{1}{2} + 2^{w-1}(\frac{1}{2} - p)^w, \tag{9}$$

using the Piling up lemma[9]. Replace the bits $(u_0, u_1, ..., u_{B-1})$ in the set (8) with a guess $\hat{\mathbf{U}} = (\hat{u}_0, \hat{u}_1, ..., \hat{u}_{B-1})$. If $(\hat{u}_0, \hat{u}_1, ..., \hat{u}_{B-1}) \neq (u_0, u_1, ..., u_{B-1})$, (that is, if one or more of the guessed bits are wrong) each equation will hold with a probability $P = 0.5$. If the guess is right, each equation will hold with a probability $P_w > 0.5$. We see that the (N, l) block-code is reduced to a (m, B) block-code, and the decoding problem is to decode message blocks of length B that are sent as codewords of length m through a channel with crossover probability $1 - P_w$.

The decoding can be done the following way. For all the 2^B possible guesses for $\hat{\mathbf{U}} = (\hat{u}_0, \hat{u}_1, ..., \hat{u}_{B-1})$, test $\hat{\mathbf{U}}$ with all the equations in the set (8), and give the guess one point for every equation in the set that holds. Afterward, assume that $(u_0, u_1, ..., u_B) = \hat{\mathbf{U}}$ for the guess of $\hat{\mathbf{U}}$ that has the highest score. In this way we get the first B bits of the secret initialization bits $(u_0, u_1, ..., u_l)$. The procedure can be repeated to find the rest of the bits $(u_B, u_{B+1}, ..., u_{l-1})$.

The complexity for this algorithm is

$$O(2^B \cdot m) \tag{10}$$

since we have to test m equations on the 2^B different guesses of $\hat{\mathbf{U}}$.

2.4 Fast Correlation via Convolutional Codes

In [4] Johansson and Jönsson showed how the equation set (8) can be used to decode the key stream $\mathbf{z}$ via convolutional codes. The problem is formulated as decoding of a $(m, 1, B)$ convolutional code, and the decoding is done using the Viterbi algorithm. This algorithm is optimal, but has relatively high usage of memory. In convolutional codes the coding is done over T bits. Using the fact that the equations are cyclic, the algorithm in Sect. 2.3 is used for calculating the metrics for each state $\hat{\mathbf{U}}$ at time t, $0 \leq t < T$. The algorithm in Sect. 2.3 is actually a special case of the fast correlation attack via convolutional code with $T = 1$. When the metrics are calculated, we try to find the longest possible path through the states $0 \leq t < T$. We see that the problem is transformed into finding the longest path trough a $2^B \times T$ trellis. The Viterbi algorithm is optimal for solving this problem. We refer [4] for details about the convolutional attacks.

2.5 Theoretical Analysis and Complexity

In [5] Johansson and Jönsson, presented a theoretical estimate of the success rate for fast correlation attacks via convolutional codes.

For a given bit stream of length N generated by a shift register with feedback polynomial $g(x)$, the expected number of equations of type (5) is

$$E(m) = \frac{\binom{N-T-l}{w}}{2^{l-B}} \approx \frac{\binom{N}{w}}{2^{l-B}} \tag{11}$$

Let $p_e < l \cdot 2^{-B}$ and $p = P(z_t \neq u_t)$. Then the convolutional attack described in Sect. 2.4 has a success with probability $1 - p_e$ if

$$p \leq \frac{1}{2} - \frac{1}{2}\left(\frac{8ln2}{m}\right)^{\frac{1}{2w}}. \tag{12}$$

The probability p is set by the stream cipher. The closer p is to 0.5, the more equations are needed to fulfill (12). One way to find more equations is to increment w, the number of bits on the right hand side of the equations. If we do this, each equation we find gets weaker, see equation (9). But although each equation is weaker, we find so many of them that they together give more information about the unknown key bits $\mathbf{u}^I$. The problem with this is, as shown below, that the complexity of the attack also increases when we use many more equations. In Sect. 4 we will describe a new method to solve this problem.

The complexity of the convolutional attack in [4,5] is $O(2^B \cdot m \cdot T)$, since we decode over T bits. This can be rewritten using equation (11) and noting that $m = 2^B \frac{\binom{N}{w}}{2^l}$. Let $o = \frac{\binom{N}{w}}{2^l}$. In this way we see that the complexity can be formulated as

$$O(2^{2B} \cdot o \cdot T), \tag{13}$$

The complexity for the simple attack in [2] is $O(2^{2B} \cdot o)$. Using this formulation, we see that if we use all the equations for given w, N, B and l, the run time complexity increases with a factor 4, when we increment B by one.

3 Methods for Finding Equations

In this section we will describe a fast method for finding many equations. The method is in some ways similar to the solution of the generalized birthday problem that Wagner presented in [1].

We have an equation of the form (5) if we find w columns in the generator matrix G of length N that sum to zero in the last $l - B$ positions. For $w = 2$ we sort the N columns from the generator matrix. Equal columns will then be located next to each other. The complexity of this method is $\mathrm{O}(N \log N)$.

3.1 A Basic Method for Finding Equations with $w > 2$

We will now describe a simple and well known algorithm for finding equations when $w > 2$.

First we sort the columns in the $l \times N$ generator matrix G according to the values in last $l-B$ bits. Then we run through all possible sums of $w-1$ columns, and search for columns in G that are equal to the sums in the last $l - B$ bits. The sum of these w columns is then zero in the $l - B$ last bits. The complexity of this algorithm will be $O(N^{w-1} \log N)$.

This method is straightforward and seems good since we find all the equations. The problem is when $l - B$ becomes big, since it is less likely that the sum of the combination of $w-1$ columns matches a single column. The number of possible different values in the $l - B$ last bits are 2^{l-B}. If we pick a random combination of $w-1$ columns we will have a probability less than $P_m = N/2^{l-B}$ of getting a match from the sorted generator matrix. If $N = 2^{20}$, $B = 15$ and $l = 40$ then $P_m = 2^{-5}$, so on average each 32'th sum combination will give an equation. If we increase the degree of the feedback polynomial to $l = 60$, the probability of finding an equation for given $w - 1$ columns will be reduced to $P_m = 2^{-25}$. Since an equation with $w = 4$ is a very weak equation, we need millions of equations in most cases.

Table 1. The table shows the percentage of the total number of equations we need for a successful convolutional attack when $N = 2^{21}$, $l = 60$, $w = 4$ and $B = 20$.

p	v
0,41	0.00068%
0,43	0,0051%
0,45	0,075%
0,47	4.5%

The method above finds all the equations, but in fact we do not need all the equations for the attack to succeed. From (12) we get the equation

$$m_s = \frac{8 \ln 2}{(1-2p)^{2w}},$$

where m_s is the number of equations needed for success for a given crossover probability p. Then $v = \frac{m_s}{m}$ will give us the rate of the total number of equations m needed for a successful attack. Table 1 shows different rates needed for different attacks. The fact that we do not need all the equations indicates that we may use a fast method to find a subset of them.

3.2 A Method for Finding All the Equations with $w = 4$

The method described here works in certain situations where the parameters are small. The algorithm works as follows. In the first step we go through all the possible sums of pairs of columns in G. These sums are stored in a matrix G_2 and the indexes of the two columns in G are also stored. In the second step we sort the matrix G_2 according to the last $l - B$ bits. Then we search for the columns in G_2 that are equal in the last $l - B$ bits. In this way we get weight 4 equations of the form:

$$(\mathbf{f}_{j_1} + \mathbf{f}_{j_2})^{\mathbf{T}} = (\mathbf{g}_{i_1} + \mathbf{g}_{i_2} + \mathbf{g}_{i_3} + \mathbf{g}_{i_4})^{\mathbf{T}} = (c_0, c_1, ..., c_{B-1}, \underbrace{0, \ldots, 0}_{l-B}) \tag{14}$$

where the f_j's are columns in G_2 and the g_j's are columns in G.

By this method we will find all the equations 3 times. The reason for this is illustrated by the equation $g_{i_1} + g_{i_2} + g_{i_3} + g_{i_4} = 0 \Longleftrightarrow g_{i_1} + g_{i_2} = g_{i_3} + g_{i_4}$. Two other pairs giving the same equation is $g_{i_1} + g_{i_4} = g_{i_2} + g_{i_3}$ and $g_{i_1} + g_{i_3} = g_{i_2} + g_{i_4}$. This collisions are avoided if the pairing in the second step has a restriction. All the indexes on the left side of the equation must all be less or greater than the indexes on the right side. In this way $\frac{2}{3}$ of the equations will be thrown away, but the remaining $\frac{1}{3}$ will represent all the equations. This method will be impractical if N is big, since G_2 will have a length of $N_2 = \binom{N}{2}$.

3.3 A Fast Method for Finding a Subset of the Equations with $w = 4$

Here we will solve the problem concerning memory requirement in the algorithm presented above. Using this algorithm we are able to find all the equations, but the number of possible sums in step one is far too many. If we can reduce the size of G_2, without reducing the number of equations significantly, we have succeeded.

The algorithm is divided into two steps. In step one we find a subset of all the sums, where the pairing in step 2 only involves elements in that subset. The sum of two columns that are unequal in the last bits will never be zero. Therefore we may look for sums of pairs in step 1 where we require a certain value in the last $l - B_2$ positions. Without loss of generality we require zeroes in the last $l - B_2$ positions in G_2.

Let $B_4 < B_2 < l$. First we sort the columns in G according to the last $l - B_2$ positions. Then we go through the matrix and find the columns that sum to zero in the last $l - B_2$ positions and store them in matrix G_2. The original positions

Algorithm 1 Algorithm for finding a subset of all the equations with $w = 4$

Input: G, N, B_2, $B_4 < B_2$, l.
Step 1:
sort the $l \times N$ matrix G according to the last $l - B_2$ bits.
For $0 \leq i_1, i_1 < N$ find all pairs of columns $\mathbf{g}_{j_1}$ and $\mathbf{g}_{j_2}$ that sums to

$$\mathbf{f}_j^T = (\mathbf{g}_{i_1} + \mathbf{g}_{i_2})^T = (d_0, d_1, ..., d_{B_2-1}, \underbrace{0, \ldots, 0}_{l-B_2})$$

Add $\mathbf{f}_j$ and indexes $i_1 + i_2$ to matrix G_2.
Step 2:
sort $l \times N_2$ matrix G_2 according to the last $l - B_4$ bits.
For $0 \leq j_2, j_4 < N_2$ find all pairs of columns $\mathbf{g}_{j_1}$ and $\mathbf{g}_{j_2}$ that sums to

$$(\mathbf{f}_{j_1} + \mathbf{f}_{j_2})^T = (\mathbf{g}_{i_1} + \mathbf{g}_{i_2} + \mathbf{g}_{i_3} + \mathbf{g}_{i_4})^T = (c_0, c_1, ..., c_{B_4-1}, \underbrace{0, \ldots, 0}_{l-B_4})$$

Add $c_0, c_1, ..., c_{B-1}$ and the indexes i_1, i_2, i_3, i_4 to F
Return: F

of the columns in the sum are also stored. The size of G_2 is thereby reduced by a factor of 2^{l-B_2}. In the second step we repeat the algorithm using B_4 on G_2. We sort the matrix G_2 according to the last $l - B_4$ bits, in order to find pairs of columns from G_2, where the sum is zero in the last $l - B_4$ bits. In this way we get weight 4 equations of the form (5). The pseudo code for this is shown in Algorithm 1.

Algorithm 1 is a method which may keep the memory requirements sufficiently low. From (11) we get the size N_2 of G_2 ,

$$N_2 = \frac{\binom{N}{2}}{2^{l-B_2}} \approx \frac{N^2}{2^{l-B_2+1}}.$$

It is possible to run this algorithm several times to find even more equations. Instead of keeping the last $l - B_2$ bits zero in the first step, we may repeat this algorithm requiring these bits having the fixed values $(d_{B_2}, d_{B_2+1}, ..., d_l) \neq \mathbf{0}$. We may choose to only vary the first two bits, and run the algorithm 2^2 times. Thus we get four times as many equations compared to running it only once. The cost is that we have to sort the matrixes G and G_2

Using Algorithm 1 some of the equations we get will be equal, called collisions. If we use algorithm 1 repeatedly changing r bits (that is: we repeat 2^r times) this bound is

$$p(collision) < 2^{(B_2+r)-l} + 2^{2(B_2+r-l)} < 2 \cdot 2^{(B_2+r-l)} = 2^{(B_2+r+1-l)}$$

since $B_2 + r - l < 0$. If we do not use repetitions we set $r = 0$. In practical attacks, this probability will be very low, and the simulations show that this has little impact on the decoding.

4 Fast Decoding Using Quick Metric

In Sect. 3 we presented a fast method for finding a huge number of equations. These equations can give us a lot of information about the initialization bits. But since there are so many of them, we get two new problems. It will take too much memory to store all the equations, and the complexity will be too high when we use them to calculate the metrics during decoding. Thus, we need an efficient method for storing the equations, and an efficient method for using them.

The complexity for calculating the metrics by the method in Sect. 2.3, is $O(2^B \cdot m)$, where m is the number of equations and B is the message block size of the code. If m is very high, the decoding problem can be to complex. We reduce the decoding complexity to $O(2^{2B} + m)$ by the following two methods referred to as *Quick Metric*.

4.1 A New and Efficient Method for Storing the Equations

Let $m \gg 2^B$ be the number of equations found using the method described in Sect. 3 with $B = B_4$. We get an equation set like (8). The main observation here is that although there are m different equations, there exist only 2^B different versions of the left side of the equations. This means that many equations will share the same left sides defined by $(c_0, c_1, ..., c_{B-1})$ when $m \gg 2^B$. We can now use *counting sort* to store the equations. Let E be an integer array of size 2^B. When an equation of the form (5) is found, we set

$$e = c_0 + 2c_1 + 2^2c_2 + ... + 2^{B-1}c_{B-1}. \tag{15}$$

Then we count the equation by setting $E(e) \leftarrow E(e) + 1$.

At this point we have stored the left side of the equation. To store the right side, we use another integer array, $sum()$, of size 2^B. Then we calculate the binary sum $s = (z_{i_1} + z_{i_1} + ... + z_{i_w}) \bmod 2$ for the given $(i_1, i_2, ..., i_w)$. Finally we set $Sum(e) \leftarrow Sum(e) + s$.

When the search for equations is finished, $E(e)$ is the number of the equations of type e that was found, and $Sum(e)$ is the number of equations of type e that sum to 1 on the right hand side for a given key stream $\mathbf{z}$.

Algorithm 2 shows a pseudo code for this idea. Here the idea is expanded so that it works with decoding via convolutional codes as presented in [4,5]. We assume that the search methods in Algorithm 1 are used to find the w columns that sum to zero in the last B bits. When decoding is done via convolutional codes, the equations are cycled T times when we decode over T bits. This means that we have to calculate $Sum(e)$ for every $0 \leq t < T$, since the right side of (7) is not cyclic itself. From this we get the 2-dimensional array $Sum(e, t)$. One little detail to make this work with convolutional codes, is that the bit c_B in the sum of the columns has to be 1. But this has no impact on the complexity for the algorithm.

Algorithm 2 Algorithm for storing equations (first step)

Input: G, N, T, B,w and z.
For every $i_1, i_2, ..., i_w$, $T \leq i_1, i_2, ..., i_w < N - T$,
If the columns $\mathbf{g}_{i_1}, \mathbf{g}_{i_2}, ..., \mathbf{g}_{i_w}$ in G summarize to

$$(\mathbf{g}_{i_1} + \mathbf{g}_{i_2} + ... + \mathbf{g}_{i_w})^{\mathbf{T}} = (c_0, c_1, ..., c_{B-1}, \underbrace{1, 0, \ldots, 0}_{l-B})$$

Let e be the integer value of the bits $(c_0, c_1, \ldots, c_{B-1})$.
$E(e) \leftarrow E(e) + 1$
For every t, $1 \leq t \leq T$,
$Sum(e, t) \leftarrow Sum(e, t) + (z_{t+i} + z_{t+j} + z_{t+k} \bmod 2)$
Return: The integer arrays Sum and E

4.2 A New and Efficient Method for Calculating the Metrics

Assume we have done the search for equations according to Sect. 3.3 and Algorithm 2. After this preprocessing step, we have the two arrays E and sum. Let $m_e = E(e)$ be the number of equations found of type e. Now we can test m_e equations on a guess $\hat{\mathbf{U}}$ in just one step instead of m_e .

Make a guess $\hat{\mathbf{U}}$ for $\mathbf{u}^I$. For every equation type e, do as follows: If the sum $c_0\hat{u}_0 + c_1\hat{u}_1 + ... + c_{B-1}\hat{u}_{B-1}$ corresponding to the equation type e (see equation 15) is 1, the number of the $m_e = E(e)$ equations that hold is $sum(e)$. The metric for the guess $\hat{\mathbf{U}}$ is incremented with $sum(e)$. If $c_0\hat{u}_0 + c_1\hat{u}_1 + ... + c_{B-1}\hat{u}_{B-1} = 0$, the number of the m_e equations that hold is $m_e - sum(e)$, and the metric is incremented with $m_e - sum(e)$. Algorithm 3 shows the pseudo code for this idea.

Now we have calculated the metric for one guess in just 2^B steps instead of $m > 2^B$ steps. The complexity for this part of the attack is actually independent of the amount of equations that are used, and the complexity for calculating the metrics for all the 2^B guesses is $O(2^{2B})$. The reason that the overall complexity is $O(2^{2B} + m)$, is that we have to go through all m equations once in the preprocessing, each time we want to analyze a new key stream z. Using the search algorithm in Sect. 3, we can do some processing independently from z. But in the end we have to go through m equations and save the $z_{i_1} + z_{i_2} + ... + z_{i_w}$ in array sum for each equation. This part of the search algorithm is almost linear in m.

4.3 Complexity and Properties

When we use Quick Metric, the decoding is done in two steps. The first step is the building of the equation count matrix E. The second step is decoding using the Viterbi algorithm with complexity $O(T \cdot 2^{2B})$, because of Quick Metric. The building of matrix E can be divided into 3 parts. First the sorting of G of length N, then the sorting of G_2 of length N_2. Finally we have to go through the sorted G_2 and save all the equations in E. Thus, the total complexity for the first step

Algorithm 3 Quick Metric algorithm (second step)

Input: state $\widehat{\mathbf{U}}$, time t, and the tables Sum and E .
$metric_{\widehat{\mathbf{U}}} \leftarrow 0$
For every e, $0 \leq e < 2^B$
If equation e over state $\widehat{\mathbf{U}}$ sums to 1,
$metric_{\widehat{\mathbf{U}}} \leftarrow metric_{\widehat{\mathbf{U}}} + Sum(e,t)$
Else
$metric_{\widehat{\mathbf{U}}} \leftarrow metric_{\widehat{\mathbf{U}}} + (E(e) - Sum(e,t))$
Return: $metric_{\widehat{\mathbf{U}}}$

is $O(N \cdot logN + N_2 \cdot logN_2 + T \cdot m)$. Since m has to be very high for our attack, the complexity is most often dominated by $T \cdot m$, and the overall complexity for the first step is $O(T \cdot m)$.

It will vary which of the two steps that will dominate the complexity. Thus, the total run time complexity for both step is given by

$$O(T \cdot m + T \cdot 2^{2B}).$$

To guarantee success $(99,9\%)$, the number of equations m and the convolutional memory B should satisfy equation (12) where p and l is are set by the cipher system. T must be high enough so that the algorithm converge to the right path in the trellis. $T \approx l$ is enough for most cases. The complexity for the attack in [4,5] is $O(2^{2B} \cdot o \cdot T)$, where $o = \frac{\binom{N}{w}}{2^l}$.

The first observation is that when we use Quick Metric, the computational complexity for the Viterbi algorithm is independent from the number of equations m that is used for decoding. The main difference from the attacks in [4,5] is that we just have to go through all m equations once in the first step. In [4, 5] they have to go through all the m equations for every time they test one of the 2^B states. Thus, our algorithm has a big advantage when we choose to use more than 2^B equations.

A drawback for our algorithm is that we have to do the first step every time we want to decode a new stream generated by the same system. In [4,5], they just have to do the preprocessing once for each cipher system. Therefor we have to keep the complexity in the first step as low as possible. There is actual a trade off between the two steps. When the first step takes long time, the second step takes less time, and the other way around. This means that we have to choose the parameters N, B and m carefully to get the best possible attack.

The next observation is that our algorithm is stronger for last B, since we can use many more equations. That means that we can attack a system using a last B than is possible with the attacks in [4,5]. Thus, the run time for given B, w and m goes down considerably since B has a huge impact on the complexity.

Table 2. Our attack compared to previous attacks. The generator polynomial degree l for the LFSR is 60 for all the simulations. We set $T = 60$. The * is a theoretical estimate using the success rate equation (12).

Improved convolutional attack

B	p	N	w	*Total decoding complexity*
14	**0.43**	$\mathbf{15 \cdot 10^6}$	4	$\mathbf{2^{35}}$
10	**0.43**	$\mathbf{100 \cdot 10^6}$	4	$\mathbf{2^{31}}$
16	**0.47**	$\mathbf{100 \cdot 10^6}$	4	$\mathbf{2^{39}}$
11	**0.43**	$\mathbf{40 \cdot 10^6}$	4	$\mathbf{2^{30}}$

Previous convolutional attack[5]

B	p	N	w	*Decoding complexity*
20	**0.43**	$\mathbf{100 \cdot 10^6}$	2	$\mathbf{2^{38}}$
18	**0.37**	$\mathbf{600 \cdot 10^3}$	3	$\mathbf{2^{37}}$
25*	**0.47**	$\mathbf{100 \cdot 10^6}$	2	$\mathbf{2^{48}}$

Previous attack through reconstruction of linear polynomials[3]

B	p	N	w	Rounds n	*Decoding complexity*
25	**0.43**	$\mathbf{40 \cdot 10^6}$	2	4	$\mathbf{2^{41.5}}$

5 Simulations

The evaluation of the attacks need some explanation. The interesting parameters of the cipher systems we attack, are the polynomial degree l and the crossover probability p. Finally we are given a key stream of length N. We want for a given high l to be able to decode a key stream $\mathbf{z}$ where the crossover probability $p = (u_i \neq z_i)$ is as near 0.5 as possible. Of course we want to use few key stream bits and low run time complexity.

To be able to compare the different attacks, we compute the complexity for decoding as the total number of times we have to test an equation on a guessed state. The complexity for the pre-computation is computed as the number of table lookups that have to be done during the search for equations. When we use Quick Metric we have 2 steps, so the overall complexity is given by the sum of the two steps.

See Table 2 for the simulation results. It is important to notice that we have programmed and tested all the attacks in the table, and the results for p come from these tests, not the theoretical estimate (12). For this purpose we used a 2.26 GHz Pentium IV with 1 gigabyte memory running on Linux. The algorithm is fast in practice and even the biggest attack ($p = 0.47$) was done in just a few hours including the search for equations.

From the table we see that our attack is best when p is close to 0.5. For $p = 0.47$ the run time complexity of our attack is dominated by the pre-computation step which is $m \cdot T \approx 2^{39}$. The parameters for this attack is $B_2 = 34$, $B = B_4 = 16$ and $m = 2^{33}$ which gives the code rate $r = 2^{-33}$. If we use the method in [4,5], the estimated run time complexity is 2^{48}.

Another attack from Johansson and Jönsson is the the fast correlation attack through reconstruction of linear polynomials[3]. This attack has lower complexity than fast correlation via convolutional codes and it uses less memory. We can apply Quick Metric on the reconstruction algorithm, but unfortunately this will not give a better result than using it on the convolutional code attack. The reason for this is that in each round in the algorithm we would have to repeat the search for equations. To keep B sufficient low, we would have to use many rounds. Thus, the computational complexity for this would become too high.

But when we use Quick Metric on the convolutional attack, the attack achieves in most cases a much lower run time complexity than the attack in [3]. This is shown by the two attacks in Table 2 using $N = 40 \cdot 10^6$.

6 Conclusion

We have presented a new method for calculating the metrics in fast correlation attacks. This method enable us to handle the huge number of parity check equations we get when we use $w = 4$ and the method in Sect. 3. Earlier it has only been possible to handle convolutional code rates down to around $r = 1/2^{14}$. Using our method we have decoded convolutional codes with rates down to $1/2^{32}$ in just a few hours. Because of this we have done attacks on cipher systems with higher crossover probability p than before.

An open problem is the search for equations with $w = 3$. We use $w = 4$ since there exists a fast method for finding those equations. But the equations with $w = 4$ are weak, and this gives the first step high complexity. A good solution would be to use $w = 3$, with a fast search algorithm.

References

1. David Wagner, "A Generalized Birthday Problem", *CRYPTO* 2002, LNCS 2442, Springer-Verlag, 2002, pp. 288–303.
2. V. Chepyzhov, T. Johansson, and B. Smeets, "A simple algorithm for fast correlation attacks on stream ciphers", *Fast Software Encryption,* FSE'*2000,* Lecture notes in Computer Science, Springer-Verlag, 2001, pp. 181–195.
3. T. Johansson, F. Jönsson, "Fast correlation attacks through reconstruction of linear polynomials", *Advances in Cryptology*-CRYPTO'2000, Lecture Notes in Computer Science, vol. 1880, Springer- Verlag, 2000, pp. 300–315.
4. T. Johansson, and F. Jönsson, "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.

5. T. Johansson, and F. Jönsson, "Theoretical Analysis of a Correlation Attack based on Convolutional Codes", *Proceedings of 2000 IEEE International Symposium on Information Theory*, IEEE, 2000, pp. 212.
6. T. Siegenthaler, "Decrypting a class of stream ciphers using ciphertext only", *IEEE Trans. on Computers,* vol. C-34, 1985, pp. 81–85.
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. M.Matsui, "Linear Cryptanalysis Method for DES Cipher", *Advances in Cryptology*-EUROCRYPT'93, LNCS 765, 1994, pp. 386–397.