International Journal of Mathematical Archive-5(7), 2014, 205-212
Available online through www.ijma.info ISSN 2229 – 5046

AN EFFICIENT METHOD FOR CONVOLUTIONAL CODES BASED FAST CORRELATION ATTACK

Amar Pandey*[1], Manoj Kumar Singh[2] and S. S. Bedi[2]

[1]J. K. Institute of Applied Physics & Technology,
Department of Electronics & Communication, University of Allahabad, Allahabad, India.

[2]SAG, DRDO, Metcalfe House, Delhi-54, India.

(Received On: 30-06-14; Revised & Accepted On: 15-07-14)

ABSTRACT

Havard Molland et.al. Proposed efficient algorithm for fast correlation attack based on convolutional codes. Their attack had run time complexity of 2^{39} operations with weight $w = 4$ equations for LFSR of length 60. In this paper we improve Havard Molland et. al. method for weight $w = 4$. The run time complexity of our algorithm is 2^{35}. We also improve the data complexity for the attack. The results show that our algorithm gives better performance when error probability p is close to 0.5.

We also apply our method for finding weight 3 parity-check equations for LFSR of size 72 bits on a cluster machine. Simulation results for the fast correlation attack using these equations are discussed. Correlation attacks for LFSR of such a size have not been reported before.

Keywords: *LFSR, Viterbi decoding, Convolutional code, Correlation attack, Computational complexity.*

1. INTRODUCTION

One important family of encryption methods is stream ciphers. The principle behind stream ciphers is to consider the message as a stream of message symbols and encrypt each symbol individually. Let $m = m_1, m_2, \ldots\ldots$ be the plaintext of length N, to encrypt the stream cipher takes a secret key K and produces a key stream $z = z_1, z_2, \ldots\ldots z_N$. The encryption is then given as $c_t = e(z_t, m_t)$. Often, the message, the cipher text, and the key stream are sequences of binary digits and the operation e is typically $\oplus$ (XOR operation). A binary (synchronous) stream cipher where the encryption function is XOR, i.e., $c_t = (m_t \oplus z_t)$, is also called binary additive stream cipher (shown in figure).

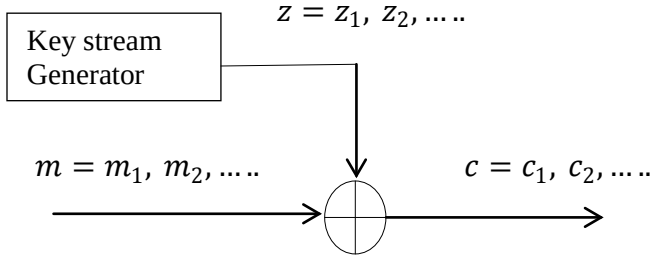

Figure: Additive Stream Ciphers.

A common method to build a keystream generator is to combine several linear feedback shift registers to get a keystream with desired statistical properties. Furthermore, the secret key K is the initial states of the LFSRs. An important class of attacks on LFSR-based stream ciphers is correlation attacks. Correlation attacks are widely applicable to stream ciphers, especially to designs based on feedback shift registers.

Corresponding author: Amar Pandey*[1]

A correlation attack exploits weakness in the combining function that allows some information about the initial state to be extracted from the output key stream. In this situation, a statistical correlation exists between the internal states and the output key stream. The first algorithm for implementing correlation attacks against a nonlinear combination generator was introduced by Siegenthaler [14, 15].

Consider the situation where the nonlinear combination generator has n LFSRs and their lengths are $l_1, l_2, \dots \dots l_n$. Given the output keystream z_t, $0 \leq t < N$, an adversary tries to determine the initial state of n LFSRs by exhaustive search. Generally, the adversary needs to guess $2^{(l_1+l_2+\cdots \dots l_n)}$ possible initial states of the LFSRs & for each guess compare generated sequence with given output sequence. However, if the correlation between the input and the output keystream of each LFSR stream can be found–which is the internal output stream of an LFSR, the adversary can test each LFSR separately and determine the initial states of each LFSR. The adversary guesses the initial state of the first LFSR and produces its LFSR stream. He then XORs the LFSR stream and the observed output keystream, and notes the number of zeros in the XORed sequence which is the same as the number of matches between LFSR sequence and the output sequence. If the initial guess is correct, the relative frequency of zeros will be close to the number of matches in obtained when last column of the truth table is compared with the column corresponding to the input LFSR. Otherwise, the XORed sequence does not present any correlation and behaves like a random string. Hence, the adversary can determine the correct initial state of the first LFSR by this statistical test. He can analyze the remaining LFSRs by the same method. Thus, instead of trying to find the initial states of all the LFSRs simultaneously, the initial states of each LFSR can be identified by using their correlations. In this situation the adversary needs only $(2^{l_1} + 2^{l_2} + \cdots \dots . + 2^{l_n})$ sequence generation & comparison steps. This approach is called a divide-and-conquer attack. Siegenthaler also showed that there was a trade-off between the degree of the combiner function and the order of correlation immunity; greater correlation immunity meant a reduced lower algebraic degree and thus lower linear complexity. To overcome this trade-off, Rueppel suggested using combiners with memory [13]. It was shown that maximum-order correlation immunity can be achieved, regardless of the linear complexity by using only one bit of memory.

Meier and Staffelbach improved the correlation attack, in their fast correlation attack [7, 8]. This type of attack does not require the search of all the initial states of the target register. Rather, it interprets system output as a noisy version if the linear code defined by LFSR and tries to "decode", the output sequence to obtain the LFSR sequence & thus the initial state. The fast correlation attack can be quite successful if the feedback polynomial has few taps, or a multiple of the feedback polynomial has a low weight [1, 9, 10, 12]. This work was followed by several papers, providing minor improvements to the initial results of Meier and Staffelbach. 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, a general thumb rule method when constructing stream ciphers that the generator polynomial or its small multiples should not be of low weight.

Thomas Johansson and Fredrik Jonsson's attack works for LFSRs with many taps. An efficient algorithm is used to find parity equations that are suitable for convolutional codes [2, 3, 4, 6]. Its performance and complexity are similar to other correlation attacks. However, a large memory requirement is still a limitation of these attacks. The decoding is done using the Viterbi algorithm, which is maximum likelihood decoding algorithm for convolutional codes. In [11], Havard Molland, John Erik Mathiassen and Tor Helleseth proposed the quick metrics for use in decoding step of fast correlation attacks, instead of using relatively few strong parity check equations and give an algorithm that has low run time complexity by using many weak parity check equations. They present a new method that is capable of performing an efficient ML decoding even when the code rate is very low and method gives the run time complexity is very low. We improve on Havard Molland method.

The paper is organized as follows. In section 2 we give some preliminaries for finding parity equations with weight is greater than two and also describe quick metrics of Molland *et al.* In section 3 we give a faster method for finding parity check equations to reduce the run time complexity. In section 4 we give some simulation results and compare the previous convolutional attacks. In section 5 we conclude with some possible extensions.

2. PRELIMINARIES

Let us start with the linear code C stemming from the LFSR sequences. There is a corresponding $l \times N$ systematic generator matrix $G = (I_l \ Z)$. Let g_i be the i^{th} column of G. Clearly, $v_i = v^l g_i$, where v^l is the initial state of the LFSR. Fix the memory of the convolutional encoder to B. If we can find w columns in the generator matrix G such that the sum of last $l - B - 1$ bits are zero and bit $l - B$ is 1 i.e.

$$\left(g_{i_1} + g_{i_2} + \cdots \dots + g_{i_w}\right)^T = \left(c_0, c_1, \dots \dots, c_{B-1}, \underbrace{0, 0, \dots \dots, 0}_{(l-B-1)}\right) \quad (1)$$

For a given B, $0 < B \leq l$, and $l \leq i_1, i_2, \ldots\ldots, i_w < N$, we get an equation form

$$c_0 v_0 + c_1 v_1 + \cdots \ldots + c_{B-1} v_{B-1} + v_B = v_{i_1} + v_{i_2} + \cdots \ldots + v_{i_w} \quad (2)$$

Observe that i^{th} Column in G gives the initialization bits $v^I = (v_0, v_1, \ldots\ldots, v_{l-1})$ that sum up to the bit v_i in the sequence v. When sum of two columns, say sum of i^{th} and j^{th} columns is zero in the last $l - B - 1$ entries ($v_{B+1}, \ldots\ldots, v_{l-1}$) then the sum $v_i + v_j$ is independent of $(v_{B+1}, \ldots\ldots, v_{l-1})$. Using equation (2), we are interested in finding m parity check equations having linear combinations of w columns of G on right side of the equations and left hand side having v_i of first $B + 1$ bits.

Replacing its $v_i s$ on RHS by corresponding key stream bits z. We get the equation set,

$$\begin{aligned}
&c_{0,0} v_0 + c_{0,1} v_1 + \cdots + c_{0,B-1} v_{B-1} + v_B \approx z_{i_{1,1}} + z_{i_{1,2}} + \cdots + z_{i_{1,w}} \\
&c_{1,0} v_0 + c_{1,1} v_1 + \cdots + c_{1,B-1} v_{B-1} + v_B \approx z_{i_{2,1}} + z_{i_{2,2}} + \cdots + z_{i_{2,w}} \\
&\qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&\qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&c_{m,0} v_0 + c_{m,1} v_1 + \cdots + c_{m,B-1} v_{B-1} + v_B \approx z_{i_{m,1}} + z_{i_{m,2}} + \cdots + z_{i_{m,w}}
\end{aligned} \quad (3)$$

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

A naive decoding approach would be guess values of first $B + 1$ bits $(v_0, v_1, \ldots\ldots, v_{B-1})$. For all the 2^{B+1} possible guesses for $(v_0, v_1, \ldots\ldots, v_{B-1}, v_B)$, test the guess, with all equations in the set (3) and give the guess one point for every equation in the set that holds. We count the score for the every adversary guesses of $(v_0, v_1, \ldots\ldots, v_{B-1}, v_B)$, and declare the one with the highest score as the most probable one. In this we get the first $B + 1$ bits of the secret initialization bits $(v_0, v_1, \ldots\ldots, v_l)$. The above method can be repeated to find the rest of the bits $(v_{B+1}, \ldots\ldots, v_{l-1})$. The complexity for this algorithm is $O(2^{B+1}.m)$[1]. In [11, 14] Johansson and Jonsson, presented a theoretical estimate of the success rate for fast correlation attacks via convolutional codes. The complexity of the convolutional attack is $O(2^m.m.T)$, since we decode over T bits.

Finding equations with $w = 2$ is easily done if we sort the columns of G based on values in the last $l - B - 1$ rows. Any pair of columns with same value in the last $l - B - 1$ rows and a different value at $l - B$ position will contribute to a parity check. The probability of getting such a pair of rows is $2^{-(l-B)}$. For a given B, as register length l increases, the probability of getting such pairs becomes exponentially low. Thus it is required to find parity checks with $w > 3$ even though the probability of such parity checks holding for the final sequence is lower.

2.1 Method for Finding Equations with weight $w > 2$

Using birthday technique [16], we will describe the method for finding many equations. Let us consider, G be a generator matrix of the size $l \times N$. Firstly we sort the generator matrix G column wise along with the last $l - B - 1$ bits. Take sum of $w - 1$ columns of G & search for columns of G that are equal to the sum in the last $l - B - 1$ bits. The sum of these w columns is then zero in the last $l - B - 1$ bits. If bit $l - B$ of the sum is 1, we use it as a equation for viterbi decoding based attack.

For weight $w = 4$, a more efficient method can be used–

Step-1. Consider all possible pairs of columns in the generator matrix G. The sums and indexes of the two columns in G are stored in the another matrix G_2.

Step-2. We sort the matrix G_2 is column wise along with the last $l - B - 1$ bits. After sorting, search for pairs of column in matrix G_2 that sum to zero in the last $l - B$ bits. We get, equation of the form (for weight 4)-

$$\left(f_{i_1} + f_{i_2}\right)^T = \left(g_{i_1} + g_{i_2} + g_{i_3} + g_{i_4}\right)^T = \left(c_0, c_1, \ldots\ldots, c_{B-1}, c_B \ \underbrace{0, 0, \ldots\ldots, 0}_{(l-B-1)}\right)$$

where the $f_j^{'s}$ are columns in G_2 and the $g_j^{'s}$ are columns in G. From these we use only those equations which have $c_B = 1$ for the viterbi decoding based attack. In Step-1 and Step-2, the number of possible sums is far too many. We can reduce the size of matrix G_2, without reducing the number of equations in Step-1'.

Step-1. We sort the generator matrix G column wise along with the last $l - B_2$ positions, where $B_2 < B$. After sorting, search for column in matrix G_2 that sum to zero in the last $l - B_2$ bits. These sums and indexes of the two columns in G are stored in the matrix G_2, then the size of matrix G_2will be $l \times N_2$, where N_2 is much smaller than $N(N-1)/2$. We get, equation of the form-

$$f_j{}^T = \left(g_{i_1} + g_{i_2}\right)^T = \left(c_0, c_1, \dots\dots, c_{B_2-1}, \underbrace{0, 0, \dots\dots, 0}_{(l-B_2)}\right)$$

The number of columns of matrix G_2 is reduced by a factor of 2^{l-B_2}.

Step-2. Repeat the Step-1' using B_4 on matrix G_2 of size $l \times N_2$, where $B_4 < B_2 < l$. We get, equation of the form is–

$$\left(f_{i_1} + f_{i_2}\right)^T = \left(g_{i_1} + g_{i_2} + g_{i_3} + g_{i_4}\right)^T = \left(c_0, c_1, \dots\dots, c_{B_4-1}, \underbrace{0, 0, \dots\dots, 0}_{(l-B_4)}\right)$$

In this way we get weight 4 equation of the form (2). Again we select those equations for which c_{B_4-1} is 1. The equations using the new G_2 matrix is a subset of the original set. If the number of equations is not sufficient for the attack, we can repeat Step-1' with last $l - B_2$ rows summing to a value other than 0 and then perform Step-2' on the resulting G_2 to get another set of parity equations. In fact this process can be repeated for each possible value in last $l - B_2$ bits to get the entire set of parity equations.

2.2 Quick Metric

When $w > 2$ we get a large number of parity checks. If number of checks $m > B$ then testing the equations for each guess of B bits or at each step in Viterbi decoding will be very expensive. In [11], Havard Molland, John Erik Mathiassen and Tor Helleseth used quick metric during decoding stage of Fast Correlation Attack which is described in the following paragraphs.

Let $m \gg 2^B$ be the number of equations found using the above method. Group equations in to sets which share the same left side i.e. which have same value for the coefficients $(c_0, c_1, \dots\dots, c_{B-1})$, We set

$$e = c_0 + 2c_1 + 2^2c_2 + \cdots \dots + 2^{B-1}c_{B-1} \tag{4}$$

Now we will count and store the number of equations of the type e, denoted by $E(e)$ i.e. $E(e)$ is the number of equations of type e. For the given keystream, find the number of equations of type e that sum to 1 on the right hand side, and denote it as $Sum(e)$. Since we have to calculate $Sum(e)$ for every time step t, $0 \le t < T$, we use a 2–dimension array $Sum(e, t)$ in the program.

Let $m_e = E(e)$ be the number of equations found of type e. Now we can test m_e equations while doing path metric computation for an edge in just one step instead of m_e steps.

For every equation of type e at time step t, if the left hand side of equation (4) corresponding to the edge is 1, since the number of the equations from this set that have same RHS is $Sum(e)$, the path metric is incremented with $Sum(e, t)$. Otherwise, metric is incremented with $m_e - Sum(e, t)$.

When we use Quick Metric, the decoding is done in two steps, firstly building of the equation count matrix Sum having complexity of $O(T.m)$ operations, second step is decoding using the Viterbi algorithm with complexity $O(T\,.\,2^{2B})$. Thus, the total run time complexity for both step is given by $O(T.m + T\,.\,2^{2B})$ [11].

3. AN IMPROVED ALGORITHM

In this section we will describe an improved method for convolutional code based fast correlation attack which was used for successful attack on a 72 stage LFSR. This algorithm for fast correlation attacks operate in two phases: In the first phase the algorithm find a set of suitable parity check equations based on the feedback taps from the LFSR and second phase uses these parity check equations in a fast decoding algorithm to recover the transmitted code word, thus the initial state of the LFSR.

3.1 Method for finding parity check equations

The method described here is for finding parity check equations when weight $w = 3$. Molland *et al.* [11] uses the sorting for finding parity check equations but here we avoid the sorting. Let $g(x)$ be the primitive polynomial of degree l for LFSR that generates the sequence v. Let α be defined by $g(\alpha) = 0$, using reduction rule we can define the generator matrix for sequence by the $l \times N$ matrix G such that $G = [\alpha^0, \alpha^1, \alpha^2, \dots\dots, \alpha^{N-1}]$.

Firstly fix the convolutional encoder size, say B. We use an additional parameter l_0, $B < l_0 < l$. We use two arrays of size 2^{l-l_0} in the program one a bit vector and other storing integers. We have to choose l_0 so that $l - l_0$ is as large as possible and yet the arrays fit in the memory.

Of the two arrays one stores if a $l - l_0$ bit value has been observed in the rows $l_0 \ldots l$ of the generator matrix G as we are scanning from left to right. The other stores the position where it was observed

In the algorithm, for every sum of 2 columns of G we select bits $l_0 \ldots l$ of the sum and check if they have been observed in the sequence. This is done using the first array with just one lookup. If so we find the position where it occurred using the second array. The corresponding column of G is generated on the fly and other bits are checked otherwise we move on to the next pair of columns.

The complexity of this algorithm will be $O(N^2)$ when l_0 is selected as suggested. The pseudo codes for the above steps are given in Algorithm 1.

For $w = 4$ also, we follow the strategy of generating the columns of the generator matrix as and when required.

3.2 Complexity

Complexity of finding parity check equations (described in section 3.1) is $O(N^2)$ and the complexity of decoding process based on Viterbi algorithm by using quick metric described in section 2.2 is $O(T \cdot 2^{2B})$.Thus, the total run time complexity is given by $O(N^2 + T \cdot 2^{2B})$.

3.3 Reduction of time complexity using MPI

Suppose we run this programme on a system with n processors, and each processor being assigned rank 0 to $n-1$. Then computations for the columns $Seq_{\propto^0}, Seq_{\propto^n}, Seq_{\propto^{2n}}$ ..... can be scheduled to run on system having rank 0. Computations for the columns $Seq_{\propto^1}, Seq_{\propto^{n+1}}, Seq_{\propto^{2n+1}}$ ..... can be scheduled to run on system having rank 1 and similarly computations for columns $Seq_{\propto^2}, Seq_{\propto^{n+2}}, Seq_{\propto^{2n+2}}$ ..... scheduled to run on system having rank 2 and so on. Then the time taken reduces by a factor of n.

This was implemented on a cluster machine and time taken (in approximate) for finding parity check equations are shown in the Table 1 with different weights.

Algorithm 1: Algorithm for finding Parity Checks of $w = 3$

```
1:  procedure FIND PARITY CHECK (f, l, N, B, l_0)
2:                                                  ▷ B < l_0 < l
3:          ▷ Notation: Seq_{a...b} denotes bits a, ..... , b − 1 of vector Seq
4:     ∝= root of f                                  ▷ Column vector
5:     Occur(i) = 0 for i ∈ 0 ..... 2^{l_0 − B}
6:     for i ∈ B + 1 ..... N do
7:          Seq = ∝^i
8:          for j ∈ i + 1 ..... N do
9:               Seq_1 = ∝^j
10:              k = (Seq ⊕ Seq_1)_{l_0 ... l}
11:              if Occur(k) == 1 then
12:                   pos = Index(k)
13:                   Seq_2 = ∝^{pos}
14:                   if (Seq_2 ⊕ Seq_1 ⊕ Seq)_{B ... l_0} == (10 ..... 0) then
15:                        Print (Seq_2 ⊕ Seq_1 ⊕ Seq)_{0 ... B}, i, j, pos
16:                   end if
17:              else
18:                   k = (Seq_1)_{l_0 ... l}
19:                   Occur(k) = 1
20:                   Index(k) = j
21:              end if
22:         end for
23:    end for
24: end procedure
```


Table 1: Timings (in approximate) for finding parity check equations					
l	B	N	w	No. of parity check equations	Time taken (in hours)
60	23	4×10^6	3	75342859	≈ 12
60	16	4×10^7	4	6505280801	≈ 10
72	23	64×10^6	3	1109747	≈ 20

4. SIMULATION RESULTS

In this section we present some simulation results for the proposed algorithm based on convolutional codes and parallel version of $w = 4$ equation construction. Time taken for cryptanalysis for weight $w = 3$ and $w = 4$. The simulation results are of the execution times of our program implemented the attack on a 256 node cluster. The runs for two different register lengths 60 & 72 and various values for p, the probability of error have been presented. In general, increasing w will increase the performance at the cost of an increased pre-computation time and increased memory requirement in pre-computation. Whereas making p close to 0.5 makes the attack more expensive in terms of time and data requirement.

The first sets of results are for the feedback polynomial of length $l = 60$ i.e. the LFSR has the following feedback polynomial,

$$g(x) = 1 + x + x^2 + x^3 + x^5 + x^6 + x^{10} + x^{11} + x^{15} + x^{17} + x^{19} + x^{20} + x^{21} + x^{25} + x^{27} + x^{34} + x^{35} + x^{36} + x^{37} + x^{38} + x^{39} + x^{41} + x^{42} + x^{45} + x^{46} + x^{47} + x^{48} + x^{49} + x^{50} + x^{51} + x^{52} + x^{53} + x^{54} + x^{56} + x^{58} + x^{59} + x^{60}$$

For a fixed size of memory of the convolutional encoder to $B = 23$, with weight $w = 3$, length of the LFSR $l = 60$ and sequence length $N = 4 \times 10^6$, the time taken for finding parity check equations are shown in Table 1. In this case $B = 23$, the improved method finds approximately $m = 75342859 \approx 2^{18.14}$ parity check equations and hence the convolutional code is of rate $R = 1/2^{18.14} \approx 2^{-18.14}$. The correct solution is found at iteration number 60 in viterbi decoding, the time taken for finding correct solutions are shown in Table 2, when probability p is varied. The total run time complexity for successful attack is $2^{35.98}$.

Table 2: Timings for $l = 60$, $B = 23$, $N = 4 \times 10^6$, weight $w = 3$	
Probability (p)	Time taken (In Second)
0.450	218.74
0.455	217.91
0.470	321.51
0.471	320.13
0.472	320.85

On fixing the memory of the convolutional encoder to $B = 16$, with weight $w = 4$, length of the LFSR $l = 60$ and $N = 4 \times 10^7$, the time taken for finding parity check equations are shown in Table 1. In this case with $B = 16$, the parallelized implementation of above method then we finds approximately $m = 6505280801 \approx 2^{22.59}$ parity check equations and hence the convolutional code is of rate $R = 1/2^{22.59} \approx 2^{-22.59}$. The correct solution is found at iteration number 60, the time taken for finding correct solutions are shown in Table 3, for various values of probability p. The total run time complexity for successful attack is 2^{35}.

Table 3: Timings for $l = 60$, $B = 16$, $N = 4 \times 10^7$, weight $w = 4$	
Probability (p)	Time taken (In Second)
0.468	10564.96
0.470	10905.36

The second set of results are for the feedback polynomial of length $l = 72$, with weight $w = 3$ i.e. the LFSR has the following feedback polynomial,

$$g(x) = 1 + x^3 + x^7 + x^8 + x^9 + x^{10} + x^{14} + x^{17} + x^{19} + x^{22} + x^{23} + x^{26} + x^{29} + x^{33} + x^{36} + x^{37} + x^{41} + x^{43} + x^{44} + x^{45} + x^{46} + x^{49} + x^{50} + x^{51} + x^{52} + x^{55} + x^{60} + x^{67} + x^{68} + x^{69} + x^{70} + x^{71} + x^{72}$$

For the simulations, we fixed the convolutional encoder memory size B to 23 with weight $w = 3$, length of the LFSR $l = 72$ and $N = 64 \times 10^6$. The time taken for finding parity check equations are shown in Table 1. We found approximately $m = 1109747 \approx 2^{13.92}$ parity check equations and hence the convolutional code is of rate $R = 1/2^{13.92} \approx 2^{-13.92}$, also the correct solution is found at step number 72. The times taken for finding correct solutions are shown in Table 4, when probability p is varied. The total run time complexity for successful attack is $2^{36.75}$. If we use the method in [4, 5], the estimated run time complexity is $2^{34.13}$.

Table 4: Timings for $l = 72$, $B = 23$, $N = 64 \times 10^6$, weight $w = 3$

Probability (p)	Time taken (In Second)
0.420	119.66
0.430	295.34
0.440	118.64
0.442	118.33
0.443	118.18
0.444	118.83
0.445	118.70
0.447	119.53
0.448	121.05
0.450	121.78

4.1 Comparison between previous convolutional attacks

We compare our attack with the fast correlation attack using convolutional codes described in [4, 5]. Havard Molland focuses on weight $w = 4$ and convolutional attack shown in Table 5. For $p = 0.47$, Molland et. al's method gives run time complexity is 2^{39} and code rate is $r = 2^{-33}$, our attack has run time complexity 2^{35} and code rate is $r = 2^{-22.59}$. Our attack is better when p is close to 0.5, because of the improvement it gives the first step complexity i.e. finding parity check equations.

Table 5: Previous convolutional attack for $l = 60$ [11]

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

5. CONCLUSION

We have improved the fast correlation attack based on convolution codes. Our method for finding parity check equations of $w = 3$ and $w = 4$ is more efficient than the method by Molland *et. al.* The run time complexity and data complexity of our attack is much lower. Further speed up of the attack is possible through distributed computing on clusters. As a part of future work we will work on parallel algorithms for the attack.

ACKNOWLEDGEMENT

The authors wish to thank Dr. P. K. Saxena for encouraging us to work on this problem. They also wish to thank Dr. N. Rajesh Pillai and Dr. Indivar Gupta for their invaluable support and informative suggestions.

REFERENCES

[1]. V. Chepyzhov and B. Smeets. On a fast correlation attack on certain stream ciphers. In Advances in Cryptology – EUROCRYPT-91, volume 547 of Lecture Notes in Computer Science, pages 176–185, February 1991.

[2]. P. Chose, A. Joux, and M. Mitton. Fast correlation attacks: An algorithmic point of view. In Advances in Cryptology – EUROCRYPT-2002, volume 2332 of Lecture Notes in Computer Science, pages 209–221. Springer, February 2004.

[3]. T. Johansson and F. Jonsson. Fast correlation attacks based on turbo code techniques. In Advances in Cryptology – CRYPTO-99, volume 1666 of Lecture Notes in Computer Science, pages 181–197.Springer, 1999.

[4]. T. Johansson and F. Jonsson. Improved fast correlation attacks on stream ciphers via convolutional codes. In Advances in Cryptology – EUROCRYPT-99, volume 1592 of Lecture Notes in Computer Science, pages 347–362. Springer, 1999.

[5]. T. Johansson and F. Jonsson. "Theoretical Analysis of a Correlation Attack based on Convolutional Codes", Proceeding of 2000 IEEE International Symposium on Information Theory, IEEE, 2000, pp. 212.

[6]. T. Johansson and F. Jonsson. Fast correlation attacks through reconstruction of linear polynomials. In Advances in Cryptology – CRYPTO-2000, volume 1880 of Lecture Notes in Computer Science, pages 300–315. Springer, 2000.

[7]. W. Meier and O. Staffelbach. Fast correlation attacks on certain stream ciphers. Journal of Cryptology, 1(3):159–176, 1989.

[8]. W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. In advances in cryptology EUROCRYPT- 88, volume LNCS 330, pages 301 –314. Springer – verlag, 1988.

[9]. M. J. Mihaljevic and J. D. Golic. A comparison of cryptanalytic principles based on iterative error-correction. In Advances in Cryptology – EUROCRYPT-91, volume 547 of Lecture Notes in Computer Science, pages 527–531. Springer, 1991.

[10]. M. J. Mihaljevic and J. D. Golic. A fast iterative algorithm for a shift register initial state reconstruction given the noisy output sequence. In Advances in Cryptology – AUSCRYPT-90, volume 453 of Lecture Notes in Computer Science, pages 165–175.Springer-Verlag, 1992.

[11]. Havard Molland, John Erik Mathiassen and Tor Helleseth, "Improved Fast Correlation Attack Using Low Rate codes", K. G. Paterson (Ed); Cryptography and coding 2003, LNCS 2898, pp. 67-81, 2003.@Springer-Verlag Heidelberg 2003.

[12]. W. T. Penzhorn. Correlation attacks on stream ciphers: Computing low-weight parity checks based on error-correcting codes. In Proceedings of the Third International Workshop on Fast Software Encryption, pages 159–172, London, UK, 1996. Springer- Verlag.

[13]. R. A. Rueppel. Design and Analysis of Stream Cipher. Springer Verlag, 1986. ISBN 0-387-16870-2.

[14]. T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. IEEE Transactions on Information Theory, IT-30:776–780, 1984.

[15]. T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only.IEEE Transactions on Computing, C 34:81–85, 1985.

[16]. David Wagner, "A Generalized Birthday Problem", CRYPTO-2002, LNCS 2442, Springer-Verlag, 2002, pp. 288-303.

Source of support: Nil, Conflict of interest: None Declared