Multi-pass Fast Correlation Attack on Stream Ciphers*

Bin Zhang and Dengguo Feng

State Key Laboratory of Information Security,
Institute of Software, Chinese Academy of Sciences,
Beijing 100080, P.R. China
martin_zhangbin@yahoo.com.cn

Abstract. Fast correlation attacks are one of the most important attacks against stream ciphers. Previous results on this topic mainly regard the initial state of the involved linear feedback shift register as a whole and only use one sort of parity-checks to decode the corresponding linear code. In this paper we propose a new kind of attack, called multi-pass fast correlation attack, on stream ciphers. This kind of attack can make good use of different kinds of parity-checks without increasing the asymptotic complexity and restore the initial state part-by-part. It has no restriction on the weight of the underlying linear feedback shift register and both theoretical analysis and simulation results show that it is more efficient than all the previously known fast correlation attacks.

Keywords: Stream cipher, Fast correlation attack, Linear feedback shift register (LFSR), Parity-check.

1 Introduction

Stream ciphers are an important class of encryption algorithms. They are widely used in many applications and a deliberately designed stream cipher is often more efficient than a block cipher in software and/or in hardware. However, the security of stream ciphers has not been widely and deeply studied as what have been done for block ciphers. It is helpful to launch new attacks on stream ciphers since the security of a cipher can only be measured by attacks.

So far, several kinds of attacks have been developed against stream ciphers, e.g. (fast) correlation attacks [2,3,4,10,11,14,16,17,19,21,23,24,26], (fast) algebraic attacks [1,6,7]. A popular and powerful approach to attack stream ciphers is to exploit different kinds of correlations between the keystream and some subset of the involved LFSRs. In a known plaintext scenario, these correlations can be used to restore the secret keys, usually the initial states of some LFSRs, from the keystream. The original idea was proposed by Siegenthaler [26] in 1984. Since then, many improvements have been made, producing more and

* Supported by the National Natural Science Foundation of China (Grant No. 90604036, 60603018, 60373047) and the National Grand Fundamental Research 973 program of China (Grant No. 2004CB318004).

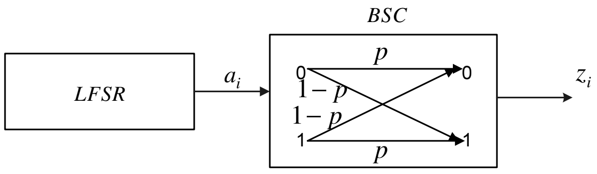

Fig. 1. Model for a fast correlation attack

more powerful fast correlation attacks [2,3,4,14,16,17,21,23,24] on LFSR-based stream ciphers. Not only the nonlinear combination generators but also the nonlinear filter generators, irregularly clocked generators [29], and many variations [5,18] are vulnerable to such a attack under proper conditions. In most articles, the corresponding cryptanalysis problem is viewed as a decoding problem as shown in Figure 1. In such a model, the output keystream $\{z_i\}$ is regarded as a noisy version of the LFSR sequence $\{a_i\}$ and the nonlinearity introduced either by nonlinear boolean function or by other methods is represented as a binary symmetric channel (BSC) with crossover probability $1 - p = 0.5 - \varepsilon$ ($\varepsilon > 0$). In theory, any decoding algorithm can be applied to recover the initial state of the target LFSR, or to restore the original LFSR sequence.

Common fast correlation attacks are classified into iterative algorithms and one-pass algorithms. In an iterative attack, each z_i is associated with a priori probability $p = 0.5 + \varepsilon$, which is updated using parity-checks. These soft-values are used to modify z_i until the BSC noise is removed. In a one-pass attack, the parity-checks are used to directly select some correct bits of $\{a_i\}$ from $\{z_i\}$. If the number of these selected bits is larger than the length of the LFSR, then the initial state can be reconstructed by simple linear algebra. As a summary, the above two approaches all aim at recovering the initial state at one time, though they follow different rules. In fact, all the known fast correlation attacks except that in [3] regard the initial state of the underlying LFSR as a whole, while the attack in [3] is just a preliminary attempt which is less efficient than some attacks targeting the whole initial state [2,4,23,24].

In this paper, we propose new techniques to mount a fast correlation attack on stream ciphers, resulting in a new kind of attack called multi-pass fast correlation attack. Oppositely to the above two approaches, we combine the divide-and-conquer idea with fast correlation attack. More precisely, we first determine *some* bits of the initial state in one pass, then find out some other bits in a second pass conditioned on both the keystream and the known bits. Following this routine, we hope to recover the whole initial state step-by-step. In addition, we propose the concept of *parity-check schedule* to cooperate with the multi-pass attack, i.e. we deploy different kinds of parity-check equations originating from the same idea in different passes. The overall asymptotic complexity of the *parity-check schedule* is not increased by constructing more than one kind of parity-checks. The theoretical analysis shows that our attack can deal with large LFSRs with correlations very close to 0.5. Comparisons with other known fast correlation attacks reveal that our algorithm achieves the best trade-off between keystream length, success probability, and complexities (time, memory and preprocessing). Experimental results confirm the high efficiency of the new attack.

The rest of this paper is organized as follows. A detailed description of the multi-pass fast correlation attack is given in Section 2 together with theoretical analysis. In Section 3, simulation results and comparisons with the best two [24,4] fast correlation attacks are provided. Finally, some conclusions are given in Section 4.

2 Multi-pass Fast Correlation Attack

We first present a outline of the proposed multi-pass fast correlation attack, then a description of the attack is given in detail with theoretical analysis. A potential application of our algorithm beyond the model shown in Figure 1 is also presented at the end of this section.

2.1 Outline of Multi-pass Fast Correlation Attack

The basic idea of the attack presented here is to launch a divide-and-conquer attack to restore the initial state. We divide the initial state $(a_0, a_1, \cdots, a_{L-1})$ of the target LFSR into several parts and plan to recover them one-by-one, as shown in (1).

$$(\underbrace{a_0, \cdots, a_{k^{(1)}-1}}_{k^{(1)}}, \underbrace{a_{k^{(1)}}, \cdots, a_{k^{(1)}+k^{(2)}-1}}_{k^{(2)}}, \underbrace{a_{k^{(1)}+k^{(2)}}, \cdots,}_{\cdots} \cdots, a_{L-1}) \tag{1}$$

We first recover the first part ($k^{(1)}$-bit length) of the initial state, then proceed to determine the second part ($k^{(2)}$-bit length) conditioned on both the first part and the keystream. In this way, we can get an efficient attack without the heavy preprocessing as required by previous attacks for efficient decoding.

Preprocessing Stage. As other fast correlation attacks, we need to pre-compute the parity-checks required by the real time processing stage. In the preprocessing stage of a multi-pass fast correlation attack, we construct various parity-check equations, as shown below.

$$\begin{aligned}
a_{i_1} \oplus a_{i_2} \oplus \cdots \oplus a_{i_{t_1}} &= \sum_{i=0}^{k^{(1)}-1} x_i^{(1)} a_i \rightharpoonup \text{first pass} \\
a_{j_1} \oplus a_{j_2} \oplus \cdots \oplus a_{j_{t_2}} \oplus \sum_{i=0}^{k^{(1)}-1} x_i'^{(1)} a_i &= \sum_{i=k^{(1)}}^{k^{(1)}+k^{(2)}-1} x_i^{(2)} a_i \rightharpoonup \text{second pass} \\
&\vdots \\
a_{l_1} \oplus a_{l_2} \oplus \cdots \oplus a_{l_{t_m}} \oplus \sum_{i=0}^{\delta-1} y_i a_i &= \sum_{i=\delta}^{\delta+k^{(m)}-1} x_i^{(m)} a_i, \rightharpoonup mth \text{ pass}
\end{aligned}$$

where $\delta = \sum_{i=1}^{m-1} k^{(i)}$ $(m \geq 2)$, $t_i \leq t_1$ (for $2 \leq i \leq m$) and $\sum_{i=0}^{\delta-1} y_i a_i$ is a known linear combination of the recovered bits. In the first pass of the processing stage, we use parity-checks of the form $a_{i_1} \oplus a_{i_2} \oplus \cdots \oplus a_{i_{t_1}} \oplus \sum_{i=0}^{k^{(1)}-1} x_i^{(1)} a_i = 0$, while in the second pass, $a_{j_1} \oplus a_{j_2} \oplus \cdots \oplus a_{j_{t_2}} \oplus \sum_{i=0}^{k^{(1)}-1} x_i'^{(1)} a_i \oplus \sum_{i=k^{(1)}}^{k^{(1)}+k^{(2)}-1} x_i^{(2)} a_i = 0$ is applied, and so on.

Definition 1. *We call such a application schedule of different parity-check equations in different passes of the attack a parity-check schedule.*

For $p < 0.55$ in Figure 1, we only use parity-check equations with $t_i \leq 4$, otherwise the folded noise will drop so close to 0.5 that an efficient decoding is impossible. For N keystream bits, if $t_i \leq 3$, we use the traditional square-root time-memory tradeoff to construct the above equations, the time and memory complexity are $O(N^{\lceil (t_i-1)/2 \rceil} \log_2 N)$ and $O(N^{\lfloor (t_i-1)/2 \rfloor})$, respectively. If $t_i = 4$, we adopt the general match-and-sort algorithm in [4] to get these equations (though the usage and meaning of these parity-checks are different in [4]). The time complexity is $O(N^2 \log_2 N)$ and the memory complexity is $O(N)$. To prepare all the parity-checks involved in the *parity-check schedule*, we have to add the complexity of constructing parity-checks for each pass together. Compared with the complexity of preparing one sort of parity-checks, the overall complexity of a *parity-check schedule* is increased only by a small factor m. In most cases of our algorithm, the number of parity-checks involved in pass $i+1$ is much less than that involved in pass i, thus the asymptotic complexity of the *parity-check schedule* is not increased.

Processing Stage. This stage consists of several passes. In each pass, we exhaustively search over the current segment of the initial state and evaluate the parity-checks to record those possible values passing the test as candidates of the corresponding bits. If the current pass is not the last one, we should pass these candidates to the next pass. The bits remain unknown after all the passes can be recovered by an exhaustive search at a small scale and a correlation check procedure.

2.2 Attack Details and Theoretical Analysis

Let $f(x)$ of degree L be the feedback polynomial of the LFSR modelled in Figure 1 and $P(a_i = z_i) = p = \frac{1}{2} + \varepsilon$, $\varepsilon > 0$. From the initial state $(a_0, a_1, \cdots, a_{L-1})$, we have $(a_0, \cdots, a_{L-1}, a_L, \cdots, a_{N-1}) = (a_0, a_1, \cdots, a_{L-1}) \cdot G$, where N is the length of sequence $\{a_i\}$ under consideration and G is a $L \times N$ matrix over $GF(2)$:

$$G = \begin{pmatrix} g_0^1 & g_1^1 & \cdots & g_{N-1}^1 \\ g_0^2 & g_1^2 & \cdots & g_{N-1}^2 \\ \vdots & \vdots & \cdots & \vdots \\ g_0^L & g_1^L & \cdots & g_{N-1}^L \end{pmatrix}.$$

Thus each a_i is a linear combination of $(a_0, a_1, \cdots, a_{L-1})$. We regard the column vector $g_i = (g_i^1, g_i^2, \cdots, g_i^L)^T$ as a random vector, then there are $\Omega^{(1)} =$

$\binom{N}{t_1}/2^{L-k^{(1)}}$ t_1-tuple column vectors $(g_{i_1}, g_{i_2}, \ldots, g_{i_{t_1}})$ satisfying $g_{i_1} \oplus g_{i_2} \oplus \cdots \oplus g_{i_{t_1}} = (x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)}, 0, \cdots, 0)^T$, where $\oplus$ is bitwise xor and $(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})$ is a $k^{(1)}$-dimension vector with $k^{(1)} < L$. For each such t_1-tuple, we have $a_{i_1} \oplus a_{i_2} \oplus \cdots \oplus a_{i_{t_1}} = \sum_{j=0}^{k^{(1)}-1} x_j^{(1)} a_j$. We rewrite it as:

$$z_{i_1} \oplus z_{i_2} \oplus \cdots \oplus z_{i_{t_1}} = \sum_{j=0}^{k^{(1)}-1} x_j^{(1)} a_j \oplus \sum_{j=1}^{t_1} e_{i_j}, \tag{2}$$

where $e_j = a_j \oplus z_j (j = i_1, \cdots, i_{t_1})$ is the corresponding random noise variable with distribution $P(e_j = 0) = \frac{1}{2} + \varepsilon$. The basic distinguisher in pass one is that if we exhaustively search all the possible values of $(a_0, a_1, \cdots, a_{k^{(1)}-1})$, then from (2), we have

$$z_{i_1} \oplus z_{i_2} \oplus \cdots \oplus z_{i_{t_1}} \oplus \sum_{j=0}^{k^{(1)}-1} x_j^{(1)} a'_j = \sum_{j=0}^{k^{(1)}-1} x_j^{(1)} \cdot (a_j \oplus a'_j) \oplus \sum_{j=1}^{t_1} e_{i_j}, \tag{3}$$

where $(a'_0, a'_1, \cdots, a'_{k^{(1)}-1})$ is the guessed value. Let $\Delta(i_1, \cdots, i_{t_1}) = \sum_{j=0}^{k^{(1)}-1} x_j^{(1)} \cdot (a_j \oplus a'_j) \oplus \sum_{j=1}^{t_1} e_{i_j}$, it is obvious that if $(a_0, a_1, \cdots, a_{k^{(1)}-1})$ is correctly guessed, we have $\Delta(i_1, \cdots, i_{t_1}) = \sum_{j=1}^{t_1} e_{i_j}$. All the e_j's are independent random variables, thus from the piling-up lemma [20],

$$q^{(1)} = P(\sum_{j=1}^{t_1} e_{i_j} = 0) = \frac{1}{2} + 2^{t_1-1}\varepsilon^{t_1} \tag{4}$$

holds. If $(a_0, a_1, \cdots, a_{k^{(1)}-1})$ is wrongly guessed, $\Delta(i_1, \cdots, i_{t_1}) = \sum_{j:a_j \oplus a'_j = 1} x_j^{(1)} \oplus \sum_{j=1}^{t_1} e_{i_j}$. Since $x_j^{(1)}$ is the xor of t_1 independent uniform distributed variables, we have $P(x_j = 0) = P(x_j = 1) = 0.5$. Hence, when $(a_0, a_1, \cdots, a_{k^{(1)}-1})$ is wrongly guessed, $\Delta(i_1, \cdots, i_{t_1})$ have the distribution $P(\Delta = 0) = 0.5$, which is quite different from that in (4). For $\Omega^{(1)}$ such equations as (3) built from all the t_1-tuples $(g_{i_1}, g_{i_2}, \ldots, g_{i_{t_1}})$, if $(a_0, a_1, \cdots, a_{k^{(1)}-1})$ is correctly guessed, $\sum_{i=1}^{\Omega^{(1)}} (\Delta(i_1, \cdots, i_{t_1}) \oplus 1)$ should follow the binomial distribution $(\Omega^{(1)}, q^{(1)})$. Otherwise, this sum should have the binomial distribution $(\Omega^{(1)}, \frac{1}{2})$, which can be used to filter out the wrong guesses of $(a_0, a_1, \cdots, a_{k^{(1)}-1})$.

To fulfill the above observations, we need to substitute z_i into the parity-check equations and evaluate them to count the number of the vanishing Δs. The straightforward method has a time complexity of $O(2^{k^{(1)}} k^{(1)} \Omega^{(1)})$, which causes an inefficient attack. Instead we proceed as follows. First regroup $\Omega^{(1)}$ parity-check equations according to the pattern of $(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})$ and define

$$h(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)}) = \sum_{(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})} (-1)^{z_{i_1} \oplus z_{i_2} \oplus \cdots \oplus z_{i_{t_1}}}$$

for all the $(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})$ patterns appearing in the $\Omega^{(1)}$ parity-check equations. If a pattern of $(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})$ does not appear in all the $\Omega^{(1)}$ equations, we define $h(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)}) = 0$ at that point. Thus we have a well-defined function $h : GF(2)^k \to \mathbb{R}$. Consider the Walsh transform of $h(x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})$, i.e.

$$\begin{aligned} H(\omega) &= \sum_{x \in GF(2)^{k^{(1)}}} h(x)(-1)^{\omega \cdot x} \\ &= \sum_{\Omega^{(1)}} (-1)^{z_{i_1} \oplus z_{i_2} \oplus \cdots \oplus z_{i_{t_1}} \oplus \sum_{j=0}^{k^{(1)}-1} \omega_j x_j^{(1)}} = \Omega_0^{(1)} - \Omega_1^{(1)}, \end{aligned} \tag{5}$$

where $\omega = (\omega_0, \omega_1, \cdots, \omega_{k^{(1)}-1})$, $x = (x_0^{(1)}, x_1^{(1)}, \cdots, x_{k^{(1)}-1}^{(1)})$, $\Omega_0^{(1)}$ and $\Omega_1^{(1)}$ are the number of 0 and 1, respectively. Note that if $\omega = (a_0, a_1, \cdots, a_{k^{(1)}-1})$, we have

$$\sum_{j=1}^{\Omega^{(1)}} (\Delta(i_1, \cdots, i_{t_1}) \oplus 1) = \frac{H(\omega) + \Omega^{(1)}}{2}. \tag{6}$$

Hence, for each guessed value $(a'_0, a'_1, \cdots, a'_{k^{(1)}-1})$, we only need to compute one value of h's Walsh transform to get the number of vanishing Δs. There are $2^{k^{(1)}}$ such guesses, which means that we need to compute all the $2^{k^{(1)}}$ values of h's Walsh transform. Thanks to the fast Walsh transform (FWT) [13,28], this can be done efficiently (at one time) in $O(2^{k^{(1)}} k^{(1)})$ time with $O(2^{k^{(1)}})$ memory. The preparation of h takes $O(\Omega^{(1)})$ time, thus the total time complexity of this step is $O(\Omega^{(1)} + 2^{k^{(1)}} k^{(1)})$. Compared with the time complexity of the former method, $O(2^{k^{(1)}} k^{(1)} \Omega^{(1)})$, this is a large improvement.

Instead of taking the guess with the largest $H(\omega)$, we put a threshold value $T^{(1)}$ of $H(\omega)$ when we make a decision, i.e. we accept all the guesses that satisfy $(H(\omega) + \Omega^{(1)})/2 \geq T^{(1)}$ at the first pass. The probability that the right guess could pass this test is (here we use the normal distribution approximation):

$$P_1^{(1)} = \sum_{i=T^{(1)}}^{\Omega^{(1)}} \binom{\Omega^{(1)}}{i} (q^{(1)})^i (1 - q^{(1)})^{\Omega^{(1)} - i} \to \int_{T^{(1)}}^{\Omega^{(1)}+0.5} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx, \tag{7}$$

and the probability that a wrong guess would pass the test is

$$P_2^{(1)} = \sum_{i=T^{(1)}}^{\Omega^{(1)}} \binom{\Omega^{(1)}}{i} (\frac{1}{2})^{\Omega^{(1)}} \to \int_{T^{(1)}}^{\Omega^{(1)}+0.5} \frac{1}{\sqrt{2\pi}\sigma'} e^{-\frac{(x-\mu')^2}{2\sigma'^2}} dx, \tag{8}$$

where $T^{(1)}$ is the threshold to be determined according to various attack requirements specified below. In (7), $\mu = \Omega^{(1)} \cdot q^{(1)}$ and $\sigma = \sqrt{\Omega^{(1)} q^{(1)} (1 - q^{(1)})}$ are the mean and the standard deviation, respectively, while in (8), $\mu' = \Omega^{(1)} \cdot \frac{1}{2}$ and $\sigma' = \frac{1}{2}\sqrt{\Omega^{(1)}}$ are the mean and the standard deviation in the random wrong guess case. If the guess $(a'_0, a'_1, \cdots, a'_{k^{(1)}-1})$ leads to $H(a'_0, a'_1, \cdots, a'_{k^{(1)}-1}) \geq T^{(1)}$, we accept it into the next pass, otherwise we filter it out right now. It is naturally expected that $P_1^{(1)}$ is very close to 1 so that the right guess could pass the test with high probability and $P_2^{(1)}$ is very small so that all the wrong guesses could be frustrated, or at least reduced to a large extent.

The introduction of the threshold $T^{(1)}$ provides large flexibility when we actually construct a fast correlation attack. Here we list four useful cases.

Case 1. $P_1^{(1)} > 0.99$ and $P_2^{(1)} < 2^{-k^{(1)}}$. The right guess will almost certainly be accepted and none of the wrong guesses could pass the test, i.e. we already restored $k^{(1)}$ bits of the initial state.
Case 2. $P_1^{(1)} > 0.99$ and $P_2^{(1)} \approx 2^{-k_1^{(1)}}$ with $k_1^{(1)} < k^{(1)}$. The right guess will pass the test with high probability together with some wrong guesses, i.e. we have a small list of candidates of the $k^{(1)}$ considered bits.
Case 3. $P_1^{(1)} < 0.99$ and $P_2^{(1)} < 2^{-k^{(1)}}$. None of the wrong guesses could pass the test, while the right guess will go through the test with some probability, i.e. we will get no candidate in some tests. In this case, we have to repeat the whole attack several times to get a high success rate.
Case 4. $P_1^{(1)} < 0.99$ and $P_2^{(1)} \approx 2^{-k_1^{(1)}}$ with $k_1^{(1)} < k^{(1)}$. We will always get some candidates in this case, though some may be wrong. In this case, we will finally get a list of candidates of the initial state, which are to be checked by correlation match.

These four cases can be applied according to different attack conditions and requirements. If Case 1 can work, then we just choose $T^{(1)}$ such that we have already recovered $k^{(1)}$ bits. If $P_1^{(1)} > 0.99$ and $P_2^{(1)} < 2^{-k^{(1)}}$ cannot be satisfied at the same time, then other cases are also helpful, e.g. if Case 2 can work, then we have already reduced the possible values of the $k^{(1)}$ bits to some extent, while we have large flexibility to construct parity-check equations used in the next pass. In this case, we can reduce the keystream length used in the next pass without a compromise of the magnitude of parity-check equations.

Once we finished the first pass, we enter the second pass to determine the next $k^{(2)}$ bits of the initial state conditioned on both the keystream and the recovered information. From the *parity-check schedule* specified in Section 2.1, the distinguisher becomes

$$\Delta(j_1, \cdots, j_{i_{t_2}}) = z_{j_1} \oplus z_{j_2} \oplus \cdots \oplus z_{j_{t_2}} \oplus \sum_{i=0}^{k^{(1)}-1} x_i'^{(1)} a_i \oplus \sum_{j=k^{(1)}}^{k^{(1)}+k^{(2)}-1} x_j^{(2)} a'_j \quad (9)$$
$$= \sum_{j=k^{(1)}}^{k^{(1)}+k^{(2)}-1} x_j^{(2)} (a_j \oplus a'_j) \oplus \sum_{j=1}^{t_2} e_{i_j},$$

where $\sum_{i=0}^{k^{(1)}-1} x_i'^{(1)} a_i$ is a known parameter and $(a'_{k^{(1)}}, \cdots, a'_{k^{(1)}+k^{(2)}-1})$ is the guessed value of the next $k^{(2)}$ bits. There are $\Omega^{(2)} = \binom{N_2}{t_2}/2^{L-k^{(1)}-k^{(2)}}$ parity-check equations used in the second pass, where $N_2 \leq N$ is the keystream length involved in the second pass. The distinguisher works in the same way as that in the first pass, i.e. we exhaustively substitute each $(a'_{k^{(1)}}, \cdots, a'_{k^{(1)}+k^{(2)}-1})$ into $\Omega^{(2)}$ parity-check equations and evaluate them to count the number of vanishing Δs. If the guess is correct, $\sum_{i=1}^{\Omega^{(2)}} (\Delta(j_1, \cdots, j_{t_2}) \oplus 1)$ should follow the binomial distribution $(\Omega^{(2)}, q^{(2)} = 0.5 + 2^{t_2-1}\varepsilon^{t_2})$, otherwise it has binomial distribution $(\Omega^{(2)}, 0.5)$. In this way, we can filter out the wrong guesses. As in the first pass, we define

$$h^{(2)}(x^{(2)}_{k^{(1)}}, \cdots, x^{(2)}_{k^{(1)}+k^{(2)}-1}) = \sum_{(x^{(2)}_{k^{(1)}}, \cdots, x^{(2)}_{k^{(1)}+k^{(2)}-1})} (-1)^{z_{j_1} \oplus \cdots \oplus z_{j_{t_2}} \oplus \sum_{i=0}^{k^{(1)}-1} x_i'^{(1)} a_i}$$

according to the pattern of $(x^{(2)}_{k^{(1)}}, \cdots, x^{(2)}_{k^{(1)}+k^{(2)}-1})$. If a pattern of $(x^{(2)}_{k^{(1)}}, \cdots, x^{(2)}_{k^{(1)}+k^{(2)}-1})$ does not appear in the $\Omega^{(2)}$ parity-check equations, let $h^{(2)} = 0$ at that point. As before, we use the fast Walsh transform of $h^{(2)}$ to compute all the $\sum_{i=1}^{\Omega^{(2)}} (\Delta(j_1, \cdots, j_{t_2}) \oplus 1) = (H^{(2)}(\omega) + \Omega^{(2)})/2$ at one time, where $H^{(2)}$ is the Walsh transform of $h^{(2)}$ and $\omega = (a'_{k^{(1)}}, \cdots, a'_{k^{(1)}+k^{(2)}-1})$.

The time and memory complexity of the second pass are $O(2^{k^{(2)}} k^{(2)} + \Omega^{(2)})$ and $O(2^{k^{(2)}})$, respectively. As in pass one, we put another threshold value $T^{(2)}$ to make a decision, i.e. if one guessed value $a'_{k^{(1)}}, \cdots, a'_{k^{(1)}+k^{(2)}-1}$ leads to $(H^{(2)}(a'_{k^{(1)}}, \cdots, a'_{k^{(1)}+k^{(2)}-1}) + \Omega^{(2)})/2 \geq T^{(2)}$, we accept it into the third pass. Otherwise discard it. In order to get an efficient attack, in the second pass as well as in the consequent passes, we only let Case 1 ($P_1^{(i)} > 0.99$ and $P_2^{(i)} < 2^{-k^{(i)}}$ for $2 \leq i \leq m$) occur, i.e. we ignore all the wrong guesses in the passes from pass two.

So far, we have recovered $k^{(1)} + k^{(2)}$ bits of the initial state. If the number of the unknown bits is still so large that the complexity of exhaustively searching over these unknown bits and the correlation check step dominates/doubles the overall complexity, we should arrange more passes to reduce the complexity. In our experiments, we usually have three or four passes to get a satisfactory low-complexity attack.

The entire description of the precessing stage of a multi-pass fast correlation attack is given below.

- **Parameter:** m, $t_1, \cdots, t_m$, $k^{(1)}, \cdots, k^{(m)}$, p
- **Input:** keystream $\{z_i\}_{i=0}^{N-1}$, feedback polynomial $f(x)$
- **Processing:**

```
1. for $i = 1, \cdots, m$ do
   Define $h^{(i)}(x^{(i)}_{k^{(i-1)}}, \cdots, x^{(i)}_{k^{(i-1)}+k^{(i)}-1}) =$
   $\sum_{(x^{(i)}_{k^{(i-1)}}, \cdots, x^{(i)}_{k^{(i-1)}+k^{(i)}-1})} (-1)^{z_{j_1} \oplus \cdots \oplus z_{j_{t_i}} \oplus \sum_{i=0}^{k^{(i-1)}-1} y'_i a_i}$, where $h^{(1)} = h$
   and for $i = 1$, $y'_i = 0$ for $0 \leq i \leq k^{(0)}$
```

```
    apply FWT to compute $\sum_{j=1}^{\Omega^{(i)}}(\Delta\oplus 1)$ for all the $2^{k^{(i)}}$ possible guesses of
    the current $k^{(i)}$-bit division, where $\Omega^{(i)}=\binom{N_i}{t_i}/2^{L-\sum_{j=1}^{i}k^{(j)}}$ and $N_i\leq N$
    is the keystream length used in pass $i$
    **if** $\sum_{j=1}^{\Omega^{(i)}}(\Delta\oplus 1)\geq T^{(i)}$ **then**
    accept the corresponding guess into pass $i+1$
    **else** discard it
    **end if**
    **if** no guess is selected **then**
    break the loop and restart the algorithm
    **end if**
    **end for**
 2. **if** $\sum_{i=1}^{m}k^{(i)}<L$ **then**
    exhaustively search over the $L-\sum_{i=1}^{m}k^{(i)}$ bits and check each value by
    running the LFSR and computing the correlation between $\{z_i\}$ and the
    generated sequence
    **end if**
```

– **Output:** the initial state $(a_0,a_1,\cdots,a_{L-1})$ or a small list of candidates

The time complexity of the processing stage consists of the complexity of each pass and the correlation check procedure if some bits were left unrecovered after pass m. If all the candidates cannot pass the correlation test, we should run the whole algorithm several times (usually three or four times in practice) to get the correct initial state. The memory complexity of the processing stage is $O(\max_{1\leq i\leq m}\max(\Omega^{(i)}\ ,2^{k^{(i)}}))$, while the time complexity (with success rate higher than 99%) varies according to the four different cases listed before.

Case 1. The time complexity is $O(\sum_{i=1}^{m}(\Omega^{(i)}+2^{k^{(i)}}k^{(i)})+2^{L-\sum_{i=1}^{m}k^{(i)}}(\frac{1}{\varepsilon^2}))$.

Case 2. The time complexity is $O(\Omega^{(1)}+2^{k^{(1)}}k^{(1)}+2^{k^{(1)}-k_1^{(1)}}\cdot(\sum_{i=2}^{m}(\Omega^{(i)}+2^{k^{(i)}}k^{(i)})+2^{L-\sum_{i=1}^{m}k^{(i)}}(\frac{1}{\varepsilon^2})))$.

Case 3. The time complexity is $O(\alpha(\sum_{i=1}^{m}(\Omega^{(i)}+2^{k^{(i)}}k^{(i)})+2^{L-\sum_{i=1}^{m}k^{(i)}}(\frac{1}{\varepsilon^2})))$, where α is the smallest integer satisfying $(1-P_1^{(1)})^{\alpha}<0.01$.

Case 4. The time complexity is $O(\alpha(\Omega^{(1)}+2^{k^{(1)}}k^{(1)}+2^{k^{(1)}-k_1^{(1)}}\cdot(\sum_{i=2}^{m}(\Omega^{(i)}+2^{k^{(i)}}k^{(i)})+2^{L-\sum_{i=1}^{m}k^{(i)}}(\frac{1}{\varepsilon^2}))))$, where α is the smallest integer satisfying $(1-P_1^{(1)})^{\alpha}<0.01$.

Remarks. It is interesting to see that our algorithm has the same time complexity, $O(\Omega^{(i)}+2^{k^{(i)}}k^{(i)})$, in each pass as that of decoding a linear $[\Omega^{(i)},k^{(i)}]$ code by the method proposed in [19]. However, It is much worth noticing that our approach is quite different from that in [19]. In [19], this decoding was done by a minimum distance rule, whereas we follow the distinguishers built according to the parity-check schedule and the basic observations of our attack have nothing to do with the codeword distance. As can be seen above, our method is more advanced which offers large flexibility. This can be seen from two aspects. First, we can achieve arbitrary success probability from 0 to 1, which is a useful property to reduce the required keystream length to a realistic range. Second, we can freely apply the four cases listed above to deal with various attack conditions. For example, if Case 2 occurs, we still can use the parity-check equations in the second pass, though we do not know the exact value of the first $k^{(1)}$ bits (instead we only reduced the possible values of these bits in the past pass).

2.3 A Potential Application Beyond Figure 1

All the above results are based on the model shown in Figure 1. As can be seen below, this is not always necessary. We can loose the model as follows, i.e. we have a sequence of variables $s_0, s_1, \cdots, s_{n-1} \in GF(2)$ such that the following equations hold.

$$\begin{cases} s_0 &= u_0^{(0)} a_0 \oplus u_1^{(0)} a_1 \oplus \cdots \oplus u_{m-1}^{(0)} a_{m-1} \oplus e_0 \\ s_1 &= u_0^{(1)} a_0 \oplus u_1^{(1)} a_1 \oplus \cdots \oplus u_{m-1}^{(1)} a_{m-1} \oplus e_1 \\ & \quad \cdots\cdots \qquad\qquad\qquad \cdots\cdots \\ s_{n-1} &= u_0^{(n-1)} a_0 \oplus u_1^{(n-1)} a_1 \oplus \cdots \oplus u_{m-1}^{(n-1)} a_{m-1} \oplus e_{n-1} \end{cases},$$

where the coefficients $u_j^{(i)}$ $(0 \leq i \leq n-1, 0 \leq j \leq m-1)$ are known parameters and e_i $(0 \leq i \leq n-1)$ are independent random variables with distribution $P(e_i = 0) = 0.5 + \varepsilon_i$ $(\varepsilon_i > 0)$. We do not care how we derive the above linear system, our aim is to recover the m variables $a_0, a_1, \cdots, a_{m-1}$ from the above linear system. Note that the model shown in Figure 1 is just a special case of the above problem, where the coefficients are derived from the generator matrix G.

To solve the above problem, we can follow a similar multi-pass routine. More precisely, we first pre-compute a system of parity-checks from the coefficient matrix $(u_j^{(i)})_{n \times m}$ as that in Section 2.1. Once we get the actual values of $s_0, s_1, \cdots,$ s_{n-1}, we proceed as a multi-pass fast correlation attack to determine the m variables $a_0, a_1, \cdots, a_{m-1}$ part-by-part. It is worth noting that the above linear system exists for a variety of stream ciphers, e.g. summation generator [9], one-level bluetooth E0 [12]. In these cases, each a_i $(0 \leq i \leq m-1)$ corresponds to an unknown bit of the initial states of the underlying LFSRs and each s_i $(0 \leq i \leq n-1)$ is some linear combination of the keystream bits. It is our future work to investigate the possibility of applying our method to these stream ciphers.

3 Experimental Results and Comparisons

To check the actual performance of our algorithm, we made experiments on a Pentium 4 processor. To facilitate the comparisons with the best known results

Table 1. Comparisons with the two best attacks previously known with success rate higher than 99%. The LFSR polynomial is $1 + x + x^3 + x^5 + x^9 + x^{11} + x^{12} + x^{17} + x^{19} + x^{21} + x^{25} + x^{27} + x^{29} + x^{32} + x^{33} + x^{38} + x^{40}$.

Attack	Noise	Keystream	Time	Memory	Pre-computation
[24]	0.469	$2^{18.61}$	$O(2^{42})$	$O(2^{16})$	$O(2^{29})$
[24]	0.490	$2^{18.46}$	$O(2^{55})$	$O(2^{25})$	$O(2^{29})$
[4]	0.469	$2^{16.29}$	$O(2^{31})$	$O(2^{25})$	$O(2^{37})$
[4]	0.490	$2^{16.29}$	$O(2^{40})$	$O(2^{35})$	$O(2^{37})$
Ours	0.469	2^{22}	$\mathbf{O(2^{24})}$	$O(2^{23})$	$\mathbf{O(2^{27})}$
Ours	0.490	2^{24}	$\mathbf{O(2^{29})}$	$O(2^{29})$	$\mathbf{O(2^{29})}$

Table 2. Comparisons with the two best attacks previously known with success rate close to 1 against a random chosen LFSR of length 89

Attack	Noise	Keystream	Time	Memory	Pre-computation
[24]	0.469	2^{38}	$O(2^{52})$	$O(2^{18})$	$O(2^{45})$
[4]	0.469	2^{28}	$O(2^{44})$	$O(2^{25})$	$O(2^{61})$
Ours	0.469	2^{32}	$\mathbf{O(2^{32})}$	$O(2^{31})$	$\mathbf{O(2^{37})}$

in other articles, we use the standard feedback polynomial of the LFSR involved in many articles.

Due to the fact that some important attack parameters are not specified in [24], the memory/pre-computation complexities of the attack in [24] listed in Table 1 and 2 are only roughly derived only according to the formulae in [24]. Table 1 shows that multi-pass fast correlation attack provides a better tradeoff between keystream length, success probability and attack complexity. We implemented the attack on the standard 40-bit LFSR with noise 0.469 in C language on a Pentium 4 processor. The parameters are chosen as follows: $m = 2$, $t_1 = t_2 = 2$, $N_1 = N = 2^{22}, N_2 = 2^{15}$ and $k^{(1)} = 20, k^{(2)} = 13$. 7 bits were left to be restored by an exhaustive search of complexity $O(2^{17})$. The preprocessing stage (it has time complexity $O(2^{27})$ and memory complexity $O(2^{22})$) lasts for a few hours, while the decoding stage takes several minutes to output the result. Compared with the preprocessing time complexity $O(2^{37})$ in [4], which takes a few days to finish the pre-computation, the gain is obvious. Besides, the memory usage in the decoding part is only $O(2^{23})$ and the success probability is higher than 99%, which make our attack more realistic. Table 1 also lists the theoretical result for the same LFSR with noise 0.49, the corresponding attack parameters are $m = 2$, $t_1 = t_2 = 2$, $N_1 = N = 2^{24}$, $N_2 = 2^{18}$ and $k^{(1)} = 22, k^{(2)} = 11$. We leave 7 bits to be recovered by an exhaustive search of complexity $O(2^{20})$. The time and memory complexity for the preprocessing stage are $O(2^{29})$ and $O(2^{24})$, respectively. Compared with the preprocessing complexity $O(2^{37})$ in [4], the gain is rather large. The memory complexity of the processing stage is $O(2^{29})$ and the success rate is higher than 99%.

Table 3. Theoretical result of a multi-pass fast correlation attack with success rate close to 1 against a random chosen LFSR of length 103

Attack	Noise	Keystream	Time	Memory	Pre-computation
Ours	0.469	2^{36}	$\mathbf{O(2^{34})}$	$O(2^{31})$	$\mathbf{O(2^{41})}$

Table 4. Theoretical result of a multi-pass fast correlation attack with success rate close to 1 against a random chosen LFSR of length 61 with noise 0.499

Attack	Noise	Keystream	Time	Memory	Pre-computation
Ours	**0.499**	2^{31}	$\mathbf{O(2^{34})}$	$O(2^{29})$	$\mathbf{O(2^{36})}$

Table 2 yields the theoretical result for a 89-bit LFSR with noise 0.469. The time complexity of our attack is very impressive and multi-pass fast correlation attack makes good use of the keystream. The attack parameters are $m = 3$, $t_1 = 3, t_2 = t_3 = 2$, $N_1 = N = 2^{32}, N_2 = N_1, N_3 = 2^{20}$ and $k^{(1)} = k^{(2)} = 26, k^{(3)} = 24$. We need another 13-bit exhaustive search of complexity $O(2^{23})$ to recover the left 13 bits. The time and memory complexity for the preprocessing stage are $O(2^{37})$ and $O(2^{32})$, respectively. Compared with the preprocessing complexity $O(2^{61})$ in [4], our attack is more realistic. The memory usage of the decoding stage is $O(2^{31})$ and the success probability is higher than 99%.

Table 3 gives the theoretical result of a multi-pass fast correlation attack against an arbitrary weight LFSR of length 103. The parameters for our algorithm are $m = 4$, $t_1 = t_2 = 3, t_3 = t_4 = 2$, $N_1 = N = 2^{36}, N_2 = N_3 = 2^{29}, N_4 = 2^{17}$ and $k^{(1)} = 29, k^{(2)} = 21, k^{(3)} = 26, k^{(4)} = 25$. We leave 2 bits to be recovered by an exhaustive search of complexity $O(2^{12})$. The time and memory complexity for the preprocessing stage are $O(2^{41})$ and $O(2^{36})$, respectively. The memory complexity of the decoding stage is $O(2^{31})$ and the attack has a success rate higher than 99%.

Table 4 gives the theoretical result of a multi-pass fast correlation attack against an arbitrary weight LFSR of length 61 with noise 0.499. The parameters are $m = 3$, $t_1 = t_2 = t_3 = 2$, $N_1 = N = 2^{31}, N_2 = 2^{25}, N_3 = 2^{19}$ and $k^{(1)} = 29, k^{(2)} = 12, k^{(3)} = 11$. We leave 9 bits to be recovered by an exhaustive search of complexity $O(2^{29})$. The time and memory complexity for the preprocessing stage are $O(2^{36})$ and $O(2^{31})$, respectively. The memory complexity of the decoding stage is $O(2^{29})$ and the success rate is higher than 99%. This example shows that our algorithm can deal with moderately large LFSRs with smaller correlations than all the previously reported results. This extends the application scope of fast correlation attacks.

From the above examples, we can see that multi-pass fast correlation attack has at least the following advantages over the past relevant attacks:

– significantly smaller processing time complexity with similar memory complexity, i.e. it has some *uniform* property in the complexity aspect.

– significantly smaller preprocessing time complexity without a compromise of the real attack complexity.
– theoretical analyzibility and flexibility.

These features guarantee that multi-pass fast correlation attack can provide a better tradeoff between keystream length, success probability and attack complexities. The impressively low complexity of this kind of attack mainly comes from the divide-and-conquer idea and the valid application of the keystream of realistic length. Besides, the attack proposed here also provides a general framework for mounting a fast correlation attack on LFSR-based stream ciphers. If we can determine the initial state segments by some method faster than that presented here, then we can get an improved multi-pass attack.

4 Conclusions

In this paper, we presented new approaches to launch a fast correlation attack on stream ciphers. The new attack enables us to analyze larger LFSRs with smaller correlations and has better trade-off between keystream length, success probability and attack complexity. Besides, we show that the new method has some potential application of efficiently solving linear systems with noisy output derived from certain stream ciphers.

References

1. Armknecht, F., Krause, M.: Algebraic Attacks on Combiners with Memory. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 162–175. Springer, Heidelberg (2003)
2. Canteaut, A., Trabbia, M.: Improved Fast Correlation Attacks using parity-check equations of weight 4 and 5. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 573–588. Springer, Heidelberg (2000)
3. Chepyzhov, V.V., Johansson, T., Smeets, B.: A Simple Algorithm for Fast Correlation Attacks on Stream Ciphers. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 181–195. Springer, Heidelberg (2001)
4. Chose, P., Joux, A., Mitton, M.: Fast Correlation Attacks: An Algorithmic Point of View. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 209–221. Springer, Heidelberg (2002)
5. Clark, A., Dawson, E., Fuller, J., Golić, J., et al.: The LILI-128 Keystream Generator. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 22–39. Springer, Heidelberg (2001)
6. Courtois, N.T., Meier, W.: Algebraic Attacks on Stream Ciphers with Linear Feedback. In: Biham, E. (ed.) Advances in Cryptology – EUROCRPYT 2003. LNCS, vol. 2656, pp. 345–359. Springer, Heidelberg (2003)
7. Courtois, N.T.: Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 176–194. Springer, Heidelberg (2003)

8. Golić, J.D.: Computation of low-weight parity-check polynomials. Electronic Letters 32(21), 1981–1982 (1996)
9. Golić, J.D., Salmasizadeh, M. (ed.): Dawson, Fast correlation attacks on the summation generator, Journal of Cryptology, Springer-Verlag, vol. 13, pp. 245–262 (2000)
10. Golić, J.D.: Iterative optimum symbol-by-symbol decoding and fast correlation attack. IEEE Trans. Inform. Theory 47, 3040–3049 (2001)
11. Golić, J.D., Hawkes, P.: Vetorial appraoch to fast correlation attacks. Designs, Codes and Cryptography 35, 5–19 (2005)
12. Golić, J.D.: Linear cryptanalysis of bluetooth stream cipher. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 238-255, pp. 51–74. Springer, Heidelberg (2002)
13. Karpovsky, M.: Finite Orthogonal Series in the Design of Diginal Devices. John Wiley and Sons, New York (1976)
14. Johansson, T., Jösson, F.: Fast Correlation Attacks based on turbo code techniques. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 181–197. Springer, Heidelberg (1999)
15. Johansson, T.: Reduced complexity correlation attacks on two clock-controlled generators. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 342–357. Springer, Heidelberg (1998)
16. Johansson, T., Jösson, F.: Improved Fast Correlation Attacks on Stream Ciphers via Convolutional Codes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 347–362. Springer, Heidelberg (1999)
17. Johansson, T., Jösson, F.: Fast Correlation Attacks through reconstruction of linear polynomals. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 300–315. Springer, Heidelberg (2000)
18. Johansson, T., Jösson, F.: A Fast Correlation Attack on LILI-128. Information Processing Letters 81, 127–132 (2002)
19. Lu, Y., Vaudenay, S.: Faster Correlation Attack on Bluetooth Keystream Generator E0. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 407–425. Springer, Heidelberg (2004)
20. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
21. Meier, W., Staffelbach, O.: Fast Correlation Attacks on certain stream ciphers. Journal of Cryptology 159–176 (1989)
22. Menezes, A.J., Van Orschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC, Boca Raton (1996)
23. Mihaljević, M., Fossorier, M.P.C., Imai, H.: A Low-complexity and High-performance Algorithm for Fast Correlation Attack. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 196–212. Springer, Heidelberg (2001)
24. Mihaljević, M., Fossorier, M.P.C., Imai, H.: Fast Correlation Attack Algorithm with listing decoding and an application. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 208–222. Springer, Heidelberg (2002)
25. Molland, H., Helleseth, T.: An Improved Correlation Attack Against Irregular Clocked and Filtered Keystream Generators. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 373–389. Springer, Heidelberg (2004)
26. Siegenthaler, T.: Decrypting a Class of Stream Ciphers using ciphertext only. IEEE Transactions on Computer C-34, 81–85 (1985)

27. Wagner, D.: A Generalized Birthday Problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–304. Springer, Heidelberg (2002)
28. Yarlagadda, R.K., Hershey, J.E.: Hadamard Matrix Analysis and Synthesis with Applications to Communications and Signal/Image Processing, pp. 17–22. Kluwer Academic, Dordrecht (1997)
29. Zhang, B., Wu, H., Feng, D., Bao, F.: A Fast Correlation attack on the shrinking generator. In: Menezes, A.J. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 72–86. Springer, Heidelberg (2005)