An Improved Fast Correlation Attack on Stream Ciphers

Bin Zhang[1,2] and Dengguo Feng[2]

[1] Laboratory of Algorithmics, Cryptology and Security,
University of Luxembourg,
6, rue Coudenhove-Kalergi, L-1359, Luxembourg
[2] State Key Laboratory of Information Security,
Institute of Software, Chinese Academy of Sciences, Beijing, 100190, China
bin.zhang@uni.lu

Abstract. At Crypto'2000, Johansson and Jönsson proposed a fast correlation attack on stream ciphers based on the Goldreich-Rubinfeld-Sudan algorithm. In this paper we show that a combination of their approach with techniques for substituting keystream and evaluating parity-checks gives us the most efficient fast correlation attack known so far. An application of the new algorithm results in the first-known near-practical key recovery attack on the shrinking generator with the parameters suggested by Krawczyk in 1994, which was verified in the 40-bit data LFSR case for which the only previously known efficient attacks were distinguishing attacks.

Keywords: Stream ciphers, Correlation attacks, Linear feedback shift register (LFSR), Shrinking generator.

1 Introduction

Fast correlation attacks are one of the most important attacks against LFSR-based stream ciphers [18]. The aim is to recover the initial state of the involved LFSR with complexity as low as possible. The earliest work dates back to [22,17], followed by a large number of algorithms [1,2,3,9,10,12,14,15,16,19,20,25]. The basic idea of a fast correlation attack is to regard the truncated keystream as the noisy version of the underlying LFSR sequence, transmitted through a binary symmetric channel (BSC) with some crossover probability, as shown in Figure 1. Thus, restoring the initial state of the LFSR is equivalent to decoding the keystream segment by some method.

Such an attack usually has two phases: in the preprocessing phase, the attacker constructs a large number of parity-checks according to the linear recurring structure of the LFSR; in the realtime phase, the attacker uses these precomputed parity-checks to decode a given keystream segment of certain length. The preprocessing phase can be done once for all, and usually takes a relatively longer time than that of the realtime processing phase. The efficiency of a fast correlation attack highly depends on the cost of the pre-computation for finding

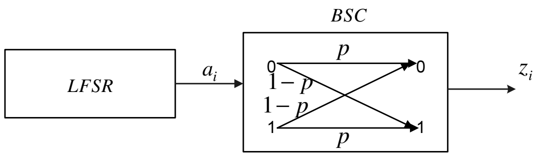

Fig. 1. Model for a fast correlation attack

desirable parity-checks. It is commonly believed that the unrealistically large preprocessing complexity is a significant barrier for decoding in the highly noisy cases. A challenging problem in this field is how to efficiently decode in these highly noisy cases with the processing complexities as low as possible, while not substantially increasing the preprocessing complexity?

In this paper, we propose a new fast correlation attack that allows us to solve this problem for noise up to 0.499 and for a LFSR of arbitrary form used in the BSC model. Our algorithm is a combination of Johansson and Jönsson's algorithm proposed at Crypto'2000 [15] with techniques for substituting keystream and evaluating parity-checks. To our knowledge, such a combination has not been studied before. We give a thorough theoretical analysis of the new algorithm, which is supported by a number of simulation results. Our algorithm can reliably decode in the highly noisy cases with considerably lower data/time/memory complexities than previously known to be possible. Besides, in all the noise and LFSR length cases considered herein, our algorithm can successfully decode without substantially increasing the preprocessing complexities. Therefore, the new algorithm is more efficient than any previously known relevant attacks and largely extends the practical application scope of fast correlation attacks.

To illustrate its power, we use the new algorithm to evaluate the security of the shrinking generator (SG) [4], which is considered as one of the strongest stream ciphers currently available. A shrinking generator consists of two LFSR's, say the data LFSR B and the control LFSR S. LFSR B is irregularly decimated by the regularly clocked LFSR S according to the following rule: the output bit of the data LFSR B is taken if and only if the current output bit of the control LFSR S is 1. For a shrinking generator with the parameters suggested by Krawczyk in [11] (LFSR B of length 61 and LFSR S of similar length), which has shown remarkable resistance against various attacks, the best known cryptanalytic results are those presented in [13] and [24]. More precisely, to restore the initial state of the data LFSR B in such a shrinking generator, the attack in [13] requires 2^{42} operations and 2^{42} keystream bits, while the attack in [24] needs 140000 keystream bits and 2^{57} operations after a pre-computation of 2^{43} operations. Both of the attacks still draw the interest of academics to this day. In contrast, given 10146 keystream bits and at most $2^{33.23}$-byte memory, our new attack against the same shrinking generator works in $2^{35.86}$ operations after a pre-computation of $2^{39.9}$ operations. This is the first-known near-practical key recovery attack against the shrinking generator with the suggested parameters in [11]. We verified our attack on the shrinking generator with a 40-bit data LFSR B on a Pentium 4 PC, in which case the only previously known efficient attacks were distinguishing attacks [5].

This paper is organized as follows. We first give a high level review of Johansson and Jönsson's algorithm [15] in Section 2. Then a description of our new algorithm is presented in Section 3 with detailed theoretical analysis. The simulation results, theoretical estimates of our algorithm and comparisons with the best previously known fast correlation attacks are provided in Section 4. The application of our algorithm to the shrinking generator with the parameters recommended by Krawczyk is described in Section 5 together with comparisons with other known attacks. Finally, some conclusions are given in Section 6.

2 Review of Johansson and Jönsson's Algorithm

Let us first specify the notations used in this paper.

- a_i is the ith output bit of the LFSR.
- z_i is the ith keystream bit.
- $P(z_i = a_i) = p = 0.5 + \varepsilon$ ($\varepsilon > 0$) is the correlation probability.
- L is the length of the LFSR.
- k ($k < L$) is the number of initial state bits to be determined.
- t is the weight of the parity-checks.
- $q = \frac{1}{2} + 2^{t-1}\varepsilon^t$ is the folded noise in a parity-check of weight t.
- N is the length of the available keystream.
- $(\cdot)^T$ is the transpose of a vector or a matrix.
- $\mathbf{b}_f \in \mathbf{F}_2^f$ is a binary column vector $\mathbf{b}_f = (b_{i_1}, b_{i_2}, \ldots, b_{i_f})^T$.
- $\Omega(\mathbf{v}_{L-k})$ is the expected number of parity-checks specified by $\mathbf{v}_{L-k}$.
- n is the number of $\mathbf{v}_{L-k}$s appearing in all the parity-checks.
- $\oplus$ is the bit-wise exclusive or.
- $\cdot$ is the inner product of two binary vectors.
- $\lceil x \rceil$ is the smallest integer greater than or equal to x.

At Crypto'2000, Johansson and Jönsson proposed a fast correlation attack on stream ciphers [15] based on the Goldreich-Rubinfeld-Sudan algorithm [6]. Although they model the decoding problem as the problem of learning a binary linear multivariate polynomial, it can be easily shown that their algorithm can also be interpreted in the BSC model in Figure 1. Hence, we use the BSC model as a unified framework hereafter.

In the preprocessing phase of Johansson and Jönsson's algorithm, the attacker constructs parity-checks of the following form:

$$\mathbf{1}_t^T \cdot \mathbf{a}_t = \mathbf{x}_k^T \cdot \mathbf{a}_k \oplus \mathbf{v}_{L-k}^T \cdot \mathbf{a}_{L-k} \ , \tag{1}$$

where $\mathbf{1}_t$ denotes the t-dimensional all-one vector, $\mathbf{a}_t = (a_{i_1}, a_{i_2}, \ldots, a_{i_t})$ (here i_j for $1 \leq j \leq t$ are arbitrary indices among the output bits), $\mathbf{a}_k = (a_0, a_1, \ldots, a_{k-1})$ and $\mathbf{a}_{L-k} = (a_k, \ldots, a_{L-1})$. In contrast with other fast correlation attacks that use parity-checks with $\mathbf{v}_{L-k} = \mathbf{0}$, in (1), $\mathbf{v}_{L-k}$ can take non-zero values. Thus, we can use several groups of parity-checks corresponding to different $\mathbf{v}_{L-k}$s here. We need not know the true value of $\mathbf{a}_{L-k}$ when determining $\mathbf{a}_k$ by evaluating the parity-checks with a *fixed* $\mathbf{v}_{L-k}$, as done in (2):

$$\mathbf{1}_t^T \cdot \mathbf{z}_t \oplus \mathbf{x}_k^T \cdot \mathbf{a}'_k = \mathbf{x}_k^T \cdot (\mathbf{a}_k \oplus \mathbf{a}'_k) \oplus \mathbf{1}_t^T \cdot \mathbf{e}_t \oplus \mathbf{v}_{L-k}^T \cdot \mathbf{a}_{L-k} \ . \tag{2}$$

In (2), $\mathbf{a}'_k$ is the guessed value of $\mathbf{a}_k$, $\mathbf{z}_t = (z_{i_1}, z_{i_2}, \dots, z_{i_t})$, $\mathbf{e}_t = (e_{i_1}, e_{i_2}, \dots, e_{i_t})$ is the random noise vector satisfying $\mathbf{z}_t = \mathbf{a}_t \oplus \mathbf{e}_t$ and $P(e_{i_j} = 0) = P(a_{i_j} = z_{i_j}) = 0.5 + \varepsilon$ for $1 \le j \le t$. Note that $\mathbf{v}_{L-k}^T \cdot \mathbf{a}_{L-k}$ is independent of $\mathbf{a}_k$ and takes either 0 or 1 in (2).

Thus, in the realtime processing phase, the attacker can evaluate the left sides of $\Omega(\mathbf{v}_{L-k})$ parity-checks like (2) and record the number of times that $\mathbf{1}_t^T \cdot \mathbf{z}_t \oplus \mathbf{x}_k^T \cdot \mathbf{a}'_k = 0$. There will exist a deviation (more or less) from $\frac{1}{2} \cdot \Omega(\mathbf{v}_{L-k})$ in the recorded number when $\mathbf{a}'_k$ is correctly guessed, and such a deviation should not be observed otherwise. To restore $\mathbf{a}_k$, the attacker sums up all the squared values of such deviations and accepts the guess resulting in the highest record as the correct one. Please see the following description of Johansson and Jönsson's algorithm.

```
Parameters: $t$, $k$, $n$
Pre-computation
    pre-compute $n$ groups of parity-checks like (1)
    with $n$ different $\mathbf{v}_{L-k}$ values
Input: keystream $\mathbf{z}_N = (z_0, z_1, \dots, z_{N-1})$
Processing
    for all the $2^k$ possible values $\mathbf{a}'_k$ of $\mathbf{a}_k$ do
      let $B_{\mathbf{a}'_k} = 0$
      for each group of parity-checks with a fixed $\mathbf{v}_{L-k}$ do
        evaluate the left side of each parity-check like (2),
        and store the total number of times that
            $\mathbf{1}_t^T \cdot \mathbf{z}_t \oplus \mathbf{x}_k^T \cdot \mathbf{a}'_k = \mathbf{0}$ as $A$
        update $B_{\mathbf{a}'_k} = B_{\mathbf{a}'_k} + (2A - \Omega(\mathbf{v}_{L-k}))^2$
      end for
      store $B_{\mathbf{a}'_k}$ in an array $U$
    end for
    search for the highest value $B^*$ in $U$ and accept the
    corresponding guess $\mathbf{a}^*_k$
Output: $\mathbf{a}_k = (a_0, a_1, \cdots, a_{k-1})$ or a small list of candidates
```

After restoring $\mathbf{a}_k$, other bits of the initial state $\mathbf{a}_L = (a_0, \dots, a_{L-1})$ can be determined with a much lower complexity, e.g. using the method in [25]. The most time-consuming step in the above algorithm is to substitute the keystream bits into the parity-checks and then to evaluate them. Since the number of the employed parity-checks is often very large and there are 2^k possible values for $\mathbf{a}_k$, this step would take a lot of time, i.e., $2^k k \sum_{\mathbf{v}_{L-k}} \Omega(\mathbf{v}_{L-k})$ operations, when it is done in the straightforward way as in [15]. This is the main bottleneck of Johansson and Jönsson's algorithm.

In the following, we will improve this algorithm by efficiently fulfilling the substitution and evaluation step. To our knowledge, such an improvement has not been studied before.

3 Our Improved Version

3.1 The Main Difference

First note that in (2), we can randomly select a value for $\mathbf{a}_{L-k}$, this does not influence the work of the parity-checks, but makes them easier to follow. Thus we have

$$\mathbf{1}_t^T \cdot \mathbf{z}_t \oplus \mathbf{x}_k^T \cdot \mathbf{a'}_k \oplus \mathbf{v}_{L-k}^T \cdot \mathbf{a''}_{L-k} = \mathbf{x}_k^T \cdot (\mathbf{a}_k \oplus \mathbf{a'}_k) \oplus \mathbf{1}_t^T \cdot \mathbf{e}_t \oplus \zeta \ , \tag{3}$$

where $\mathbf{a'}_k$ is the guessed value of $\mathbf{a}_k$, $\mathbf{a''}_{L-k}$ is the value assigned to $\mathbf{a}_{L-k}$ and $\zeta = 0$ or 1 depending on $\mathbf{a''}_{L-k}$. Other notations are the same as those in (2). In the preprocessing phase, we constructed n groups of such parity-checks, each of which is specified by a fixed $\mathbf{v}_{L-k}$ and has an expected cardinality $\Omega(\mathbf{v}_{L-k})$.

In the processing phase, the attacker evaluates the left sides of (3) and records the number of times that $\mathbf{1}_t^T \cdot \mathbf{z}_t \oplus \mathbf{x}_k^T \cdot \mathbf{a'}_k \oplus \mathbf{v}_{L-k}^T \cdot \mathbf{a''}_{L-k} = 0$. To avoid the high time complexity in the substitution and evaluation step of Johansson and Jönsson's algorithm, we proceed as follows.

For a fixed set of parity-checks specified by $\mathbf{v}_{L-k}$, group the parity-checks according to the value of $\mathbf{x}_k$ and define an integer-valued function

$$h_{\mathbf{v}_{L-k}}(\mathbf{x}_k) = \sum_{\mathbf{x}_k} (-1)^{\mathbf{1}_t^T \cdot \mathbf{z}_t \oplus \mathbf{v}_{L-k}^T \cdot \mathbf{a''}_{L-k}} \tag{4}$$

for all the coefficient vectors $\mathbf{x}_k$ appearing in this group of parity-checks. If a value of $\mathbf{x}_k$ does not appear in these parity-checks, we let $h_{\mathbf{v}_{L-k}}(\mathbf{x}_k) = 0$ in (4). Now consider the walsh transform of $h_{\mathbf{v}_{L-k}}(\mathbf{x}_k)$, i.e.,

$$H_{\mathbf{v}_{L-k}}(\omega) = \sum_{\mathbf{x}_k \in \mathbf{F}_2^k} h_{\mathbf{v}_{L-k}}(\mathbf{x}_k)(-1)^{\omega^T \cdot \mathbf{x}_k} = \sum_{\Omega(\mathbf{v}_{L-k})} (-1)^{\mathbf{1}_t^T \cdot \mathbf{z}_t \oplus \mathbf{v}_{L-k}^T \cdot \mathbf{a''}_{L-k} \oplus \omega^T \cdot \mathbf{x}_k} . \tag{5}$$

In (5), note that when $\omega = \mathbf{a'}_k$, we have $H_{\mathbf{v}_{L-k}}(\omega) = \Omega(\mathbf{v}_{L-k})_0 - \Omega(\mathbf{v}_{L-k})_1$, where $\Omega(\mathbf{v}_{L-k})_i$ is the number of i for $i = 0$ or 1. Thus given the guess $\mathbf{a'}_k$, the number of times that $\mathbf{1}_t^T \cdot \mathbf{z}_t \oplus \mathbf{x}_k^T \cdot \mathbf{a'}_k \oplus \mathbf{v}_{L-k}^T \cdot \mathbf{a''}_{L-k} = 0$ is $u(\mathbf{v}_{L-k}) = \Omega(\mathbf{v}_{L-k})_0 = \frac{\Omega(\mathbf{v}_{L-k}) + H_{\mathbf{v}_{L-k}}(\omega)}{2}$ for this group of parity-checks. For simplicity, we let $u(\mathbf{v}_{L-k}) = \frac{\Omega(\mathbf{v}_{L-k}) + |H_{\mathbf{v}_{L-k}}(\omega)|}{2}$, i.e., $u(\mathbf{v}_{L-k})$ is always greater than or equal to $\frac{\Omega(\mathbf{v}_{L-k})}{2}$.

We can use the fast Walsh transform (FWT) to *simultaneously* compute the 2^k values of $h_{\mathbf{v}_{L-k}}(\mathbf{x}_k)$'s Walsh transform function. Therefore, the total time complexity of the above substitution and evaluation step is $\sum_{\mathbf{v}_{L-k}}(2^k k+\Omega(\mathbf{v}_{L-k})(t+k))$ operations, among which $\Omega(\mathbf{v}_{L-k})(t+k)$ operations are required by the preparation of $h_{\mathbf{v}_{L-k}}(\mathbf{x}_k)$ and $2^k k$ operations are required by the FWT. The memory cost is around $c\cdot 2^k+\sum_{\mathbf{v}_{L-k}}(\ t\lceil\log_2 N\rceil+L)\Omega(\mathbf{v}_{L-k})$-bit, among which $c\cdot 2^k$ bits are the memory consumption in the FWT computation (c is a small constant and is usually ≤ 32) and $\sum_{\mathbf{v}_{L-k}}(\ t\lceil\log_2 N\rceil+L)\Omega(\mathbf{v}_{L-k})$ bits are used for the storage of parity-checks. (see Theorem 1 in Section 3.2 for an explanation). Compared to the straightforward method in [15], the gain in efficiency is obvious. To take the best of the above method and Johansson and Jönsson's idea for constructing parity-checks, we propose the following improved algorithm.

```
Parameters: $t$, $k$, $n$
Pre-computation
    pre-compute $n$ groups of parity-checks like (1)
    with $n$ different $\mathbf{v}_{L-k}$ values
Input: keystream $\mathbf{z}_N=(z_0,z_1,\ldots,z_{N-1})$
Processing
    let $B_\omega=0$ for the $2^k$ possible values of $\omega$
    for each group of parity-checks specified by $\mathbf{v}_{L-k}$ do
        let $\mathbf{a}_{L-k}$ take a randomly assigned value
        define a function $h_{\mathbf{v}_{L-k}}(\mathbf{x}_k)$ as in (4)
        apply FWT to compute $H_{\mathbf{v}_{L-k}}(\omega)$ for the $2^k$
        possible values of $\omega$
        update $B_\omega=B_\omega+(H_{\mathbf{v}_{L-k}}(\omega))^2/4$ for the $2^k$
        possible values of $\omega$
    end for
    search for $B_\omega\geq T$ and accept the corresponding
    $\omega$ as a candidate for $\mathbf{a}_k$
Output: $\mathbf{a}_k=(a_0,a_1,\cdots,a_{k-1})$ or a small list of candidates
```

Here T is the threshold determined by the success rate of the whole attack and $B_\omega=B_\omega+(H_{\mathbf{v}_{L-k}}(\omega))^2/4$ accumulates the squared biases for each guess of $\mathbf{a}_k$. The above description illustrates the structure of our improved algorithm. In practical programming, there may be some differences made for optimization.

3.2 Theoretical Analysis

Now we give a theoretical justification of our improved algorithm. First note that the expected number $\Omega(\mathbf{v}_{L-k})$ of the parity-checks with a fixed pattern $\mathbf{v}_{L-k}$ is $\frac{\binom{N}{t}}{2^{L-k}}$. Thus from (3), if $\mathbf{a'}_k$ is correctly guessed, there will exist a deviation $\Omega(\mathbf{v}_{L-k})2^{t-1}\varepsilon^t$ from $\frac{1}{2}\cdot\Omega(\mathbf{v}_{L-k})$ in the number of times that $\mathbf{1}_t^T\cdot\mathbf{z}_t\oplus\mathbf{x}_k^T\cdot\mathbf{a'}_k\oplus\mathbf{v}_{L-k}^T\cdot\mathbf{a''}_{L-k}=0$. Otherwise, such a bias should not be observed. This is the basis of our algorithm.

In our improved algorithm, the accumulated squared bias is $\sum_{\mathbf{v}_{L-k}}\big(u(\mathbf{v}_{L-k})-\frac{\Omega(\mathbf{v}_{L-k})}{2}\big)^2=\sum_{\mathbf{v}_{L-k}}\big(\frac{|H_{\mathbf{v}_{L-k}}(\omega)|+\Omega(\mathbf{v}_{L-k})}{2}-\frac{\Omega(\mathbf{v}_{L-k})}{2}\big)^2=\sum_{\mathbf{v}_{L-k}}\frac{(H_{\mathbf{v}_{L-k}}(\omega))^2}{4}$. This is just B_ω calculated in the above algorithm. Hence, we can rewrite the judgement condition $B_\omega \geq T$ in our algorithm as

$$B_\omega \geq T \Leftrightarrow \sum_{\mathbf{v}_{L-k}}\big(u(\mathbf{v}_{L-k})-\frac{\Omega(\mathbf{v}_{L-k})}{2}\big)^2 \geq T. \tag{6}$$

Obviously, when $\mathbf{a}'_k$ is correctly guessed, $u(\mathbf{v}_{L-k})$ follows the binomial distribution $(\Omega(\mathbf{v}_{L-k}), q)$, otherwise it follows the binomial distribution $(\Omega(\mathbf{v}_{L-k}), \frac{1}{2})$. If we use the normal distribution to approximate the binomial distribution, then $\sum_{\mathbf{v}_{L-k}}\big(u(\mathbf{v}_{L-k})-\frac{\Omega(\mathbf{v}_{L-k})}{2}\big)^2 \geq T$ is equivalent to

$$\frac{\Omega(\mathbf{v}_{L-k})n}{4q(1-q)} \geq \sum_{\mathbf{v}_{L-k}}\frac{(u(\mathbf{v}_{L-k})-\frac{\Omega(\mathbf{v}_{L-k})}{2})^2}{\Omega(\mathbf{v}_{L-k})q(1-q)} \geq \frac{T}{\Omega(\mathbf{v}_{L-k})q(1-q)} \tag{7}$$

when $\mathbf{a}'_k$ is correctly guessed and to

$$\Omega(\mathbf{v}_{L-k})n \geq \sum_{\mathbf{v}_{L-k}}\frac{(u(\mathbf{v}_{L-k})-\frac{\Omega(\mathbf{v}_{L-k})}{2})^2}{(\frac{1}{2}\sqrt{\Omega(\mathbf{v}_{L-k})})^2} \geq \frac{4T}{\Omega(\mathbf{v}_{L-k})} \tag{8}$$

when $\mathbf{a}'_k$ is wrongly guessed. (7) can be rewritten as

$$\frac{\Omega(\mathbf{v}_{L-k})n}{4q(1-q)} \geq \sum_{\mathbf{v}_{L-k}}\big(\frac{u(\mathbf{v}_{L-k})-\Omega(\mathbf{v}_{L-k})q+\Omega(\mathbf{v}_{L-k})\cdot 2^{t-1}\varepsilon^t}{\sqrt{\Omega(\mathbf{v}_{L-k})q(1-q)}}\big)^2 = \tag{9}$$
$$\sum_{\mathbf{v}_{L-k}}\big(\frac{u(\mathbf{v}_{L-k})-\Omega(\mathbf{v}_{L-k})q}{\sqrt{\Omega(\mathbf{v}_{L-k})q(1-q)}}+\frac{\Omega(\mathbf{v}_{L-k})2^{t-1}\varepsilon^t}{\sqrt{\Omega(\mathbf{v}_{L-k})q(1-q)}}\big)^2 \geq \frac{T}{\Omega(\mathbf{v}_{L-k})q(1-q)}.$$

Note that when $\mathbf{a}'_k$ is correctly guessed, $\frac{u(\mathbf{v}_{L-k})-\Omega(\mathbf{v}_{L-k})q}{\sqrt{\Omega(\mathbf{v}_{L-k})q(1-q)}}$ follows the standard normal distribution $\mathcal{N}(0,1)$. When $\mathbf{a}'_k$ is wrongly guessed, the variable $\frac{u(\mathbf{v}_{L-k})-\frac{\Omega(\mathbf{v}_{L-k})}{2}}{\frac{1}{2}\sqrt{\Omega(\mathbf{v}_{L-k})}}$ also follows the standard normal distribution $\mathcal{N}(0,1)$. (8) means that $\sum_{\mathbf{v}_{L-k}}\frac{(u(\mathbf{v}_{L-k})-\frac{\Omega(\mathbf{v}_{L-k})}{2})^2}{(\frac{1}{2}\sqrt{\Omega(\mathbf{v}_{L-k})})^2}$ follows the centrally chi-squared distribution when $\mathbf{a}'_k$ is wrongly guessed, while (9) implies that when $\mathbf{a}'_k$ is correctly guessed, $\sum_{\mathbf{v}_{L-k}}\frac{(u(\mathbf{v}_{L-k})-\frac{\Omega(\mathbf{v}_{L-k})}{2})^2}{\Omega(\mathbf{v}_{L-k})q(1-q)}$ follows the non-central chi-square distribution. This is the criterion used to filter out the wrong guesses.

More precisely, let $\Gamma(y)=\int_0^{+\infty}e^{-x}x^{y-1}dx$ denote the gamma function. Since there are n different $\mathbf{v}_{L-k}$s, the probability density function of a centrally chi-squared distribution is $\phi_1(x)=\frac{x^{\frac{n-2}{2}}e^{-\frac{x}{2}}}{2^{\frac{n}{2}}\Gamma(\frac{n}{2})}$ for $x>0$ and the probability density function of a non-centrally chi-squared distribution is $\phi_2(x) = \frac{e^{-\frac{(x+\delta^2)}{2}}}{2^{\frac{n}{2}}} \sum_{j=0}^{\infty} \frac{x^{j-1+\frac{n}{2}} \delta^{2j}}{\Gamma(j+\frac{n}{2}) 2^{2j} j!}$ for $x > 0$, where $\delta^2 = \sum_{\mathbf{v}_{L-k}} (\frac{\sqrt{\Omega(\mathbf{v}_{L-k})} 2^{t-1} \varepsilon^t}{\sqrt{q(1-q)}})^2$. Thus the probability that the right $\mathbf{a}_k$ could result in $B_{\mathbf{a}_k} \geq T$ is

$$P_{right} = \int_{\frac{T}{\Omega(\mathbf{v}_{L-k}) q(1-q)}}^{\frac{\Omega(\mathbf{v}_{L-k}) n}{4q(1-q)} + 0.5} \phi_2(x) dx \ , \tag{10}$$

while a wrong guess $\mathbf{a'}_k$ would pass the test with the probability

$$P_{wrong} = \int_{\frac{4T}{\Omega(\mathbf{v}_{L-k})}}^{\Omega(\mathbf{v}_{L-k}) n + 0.5} \phi_1(x) dx \ . \tag{11}$$

In our algorithm, we can control these two probabilities by carefully choosing T. A typical case is that $P_{wrong} < 2^{-k}$ with some P_{right}, i.e., none of the wrong guesses could pass the test, while the right guess could pass with some constant probability.

As a summary of the results in Section 3.1 and 3.2, we have

Theorem 1. *If the success rate of our improved algorithm is set to be P_{right} and $P_{wrong} < 2^{-k}$, then its time complexity is $\sum_{\mathbf{v}_{L-k}} (2^k k + \Omega(\mathbf{v}_{L-k})(t+k))$ operations, its memory complexity is at most $c \cdot 2^k + \sum_{\mathbf{v}_{L-k}} (t \lceil log_2 N \rceil + L) \Omega(\mathbf{v}_{L-k})$-bit and its data complexity is N-bit keystream determined by* (10)*,* (11) *and* $\Omega(\mathbf{v}_{L-k})$.

Proof. For the time complexity, note that there are n groups of parity-checks constructed, and for each group used in the processing phase, $\Omega(\mathbf{v}_{L-k})(t+k)$ operations are required by the function $h_{\mathbf{v}_{L-k}}(\mathbf{x}_k)$ and $2^k k$ operations are required by the FWT. For the memory complexity, note that we use the following straightforward way to store a parity-check of the form $\mathbf{1}_t^T \cdot \mathbf{z}_t \oplus \mathbf{x}_k^T \cdot \mathbf{a'}_k \oplus \mathbf{v}_{L-k}^T \cdot \mathbf{a''}_{L-k}$,

1. $t \cdot \lceil \log_2 N \rceil$ bits to represent the t integers $i_1, i_2, \ldots, i_t$.
2. k bits to represent the coefficient vector $\mathbf{x}_k$.
3. if $\mathbf{v}_{L-k} \neq 0$, then $L-k$ bits to represent it.

Thus, $t \lceil \log_2 N \rceil + L$ bits are needed by one parity-check. There are totally $\Omega(\mathbf{v}_{L-k})$ parity-checks in each group, so $(t \lceil \log_2 N \rceil + L) \Omega(\mathbf{v}_{L-k})$ bits are required by each group. In addition, $c \cdot 2^k$ bits are required for the computation of the FWT, where c is a small constant determined by k and by the size of a float precision floating-point number. In our analysis, $c \leq 32$ is enough for the current usage. The data complexity is determined by the success rate of the whole attack and we can use (10), (11) and $\Omega(\mathbf{v}_{L-k}) = \frac{\binom{N}{t}}{2^{L-k}}$ to determine it. $\square$

We use the traditional time/memory trade-off to pre-compute the parity-checks with a fixed pattern $\mathbf{v}_{L-k}$. That is, first compute and sort in a table the formal expression of the sum of $\lfloor \frac{t}{2} \rfloor$ a_is in the L initial state bits. Then sort the table and compute the formal sum of the other $\lceil \frac{t}{2} \rceil$ a_is. An exclusive-or collision value equal to $\mathbf{v}_{L-k}$ in the table provides us a parity-check. For the keystream of N-bit length, the time/memory complexities in the preprocessing phase are about $N^{\lceil \frac{t}{2} \rceil} \log_2 N$ operations (even $N^{\lceil \frac{t}{2} \rceil}$ operations by some hashing techniques) and $N^{\lfloor \frac{t}{2} \rfloor}(t\lceil \log_2 N \rceil + L)$-bit, respectively.

4 Simulations, Theoretical Estimates and Comparisons

In general, the simulation results of our algorithm match the theory very well. First we give some experimental results of our algorithm in the same scenarios as those considered in [15]. We only list some typical cases in Table 1[1], other cases are omitted due to limited space.

Table 1. Simulations and comparisons with the basic algorithm in [15]

attack	[15]	ours	[15]	ours	[15]	ours	[15]	ours	[15]	ours
L	40	40	40	40	60	60	60	60	60	60
p	0.64	0.64	0.55	0.55	0.57	0.57	0.68	0.68	0.6	0.6
t	2	3	2	3	2	3	3	5	2	5
N	$4 \cdot 10^5$	$\mathbf{5 \cdot 10^3}$	$4 \cdot 10^5$	$\mathbf{10^4}$	$4 \cdot 10^7$	$\mathbf{2 \cdot 10^5}$	$1.5 \cdot 10^5$	$\mathbf{8 \cdot 10^3}$	$4 \cdot 10^7$	$\mathbf{9 \cdot 10^3}$
time $\approx$	3 *min*	5 *sec*	3 *min*	25 *sec*	106 *min*	6 *min*	4.6 *min*	20 *sec*	13 *min*	1 *min*

We implemented our attack in C on a Pentium 4 PC running under windows XP. To make an accurate comparison, we also implemented one instance ($L = 40$, $p = 0.64$, $N = 4 \cdot 10^5$) of the basic algorithm in [15], the time cost is about 4 minutes (instead of 3 minutes). Although the time results of Johansson and Jönsson's algorithm listed in Table 1 are from [15] and are obtained on a different platform (Sun Ultra-80 running under Solaris), the above instance we implemented shows that the time comparisons are still meaningful. Our algorithm can successfully decode with largely reduced data complexities. For example, in the case that $L = 40$ and $p = 0.55$, our algorithm needs only 10^4 bits, while the attack in [15] needs $4 \cdot 10^5$ bits to decode. The longest pre-computation time of our algorithm in Table 1 are tens of minutes occurring in the two cases that $L = 60$ with $t = 5$. In other cases, the pre-computation costs are all negligible.

Next, we compare our algorithm to the two algorithms in [3,25], which are the best previously known fast correlation attacks. Table 2 and 3 show that our algorithm compares favorably to these two attacks.

[1] The feedback polynomial of the 40-bit LFSR used in Table 1 and 2 is $1 + x + x^3 + x^5 + x^9 + x^{11} + x^{12} + x^{17} + x^{19} + x^{21} + x^{25} + x^{27} + x^{29} + x^{32} + x^{33} + x^{38} + x^{40}$. The feedback polynomial of the 60-bit LFSR in Table 1 is not released in [15], we simply choose the polynomial $x^{60} + x + 1$ in our experiments. Since both the attack in [15] and our algorithm are applicable to arbitrary form LFSR, this choice does not influence the experimental results.

Table 2. Experimental results, theoretical predictions of our algorithm and comparisons with the best previously known attacks with success rate close to 1

p	*attack*	L	N	t	k	*time*	*memory*	*pre − computation*
0.531	[3]	40	80000	3	1	2^{31}	$2^{34.1}$	2^{37}
	[25]	40	2^{22}	2	20	2^{24}	$2^{32.8}$	2^{27}
	ours	40	**40000**	3	12	$\mathbf{2^{20}}$	$\mathbf{2^{25}}$	$\mathbf{2^{30.6}}$
	[3]	89	2^{28}	3	1	2^{44}	$2^{35.81}$	2^{61}
	[25]	89	2^{32}	3	26	2^{32}	$2^{41.12}$	2^{64}
	ours	89	$\mathbf{2^{28}}$	3	22	$\mathbf{2^{29.8}}$	$\mathbf{2^{27.3}}$	$\mathbf{2^{56}}$
0.51	[3]	40	80000	3	1	2^{40}	$2^{40.54}$	2^{37}
	[25]	40	2^{24}	2	22	2^{29}	$2^{35.13}$	2^{29}
	ours	40	**50000**	3	16	$\mathbf{2^{24.13}}$	$\mathbf{2^{26.7}}$	$\mathbf{2^{31.3}}$

Table 3. Theoretical estimates of our algorithm and comparisons with the algorithm in [25] with success rate close to 1 (the result in [25] for the case $L = 61$ and $p = 0.501$ is not correct, here we list the correct result obtained according to the formulas in [25])

p	*attack*	L	N	t	k	*time*	*memory*	*pre − computation*
0.501	[25]	61	2^{36}	2	22	2^{43}	$2^{49.71}$	2^{42}
	ours	61	$\mathbf{2^{31}}$	2	21	$\mathbf{2^{28.8}}$	$\mathbf{2^{31.3}}$	$\mathbf{2^{39.3}}$
0.531	[25]	103	2^{36}	3	29	2^{34}	$2^{38.51}$	2^{72}
	ours	103	$\mathbf{2^{32}}$	3	24	$\mathbf{2^{31.6}}$	$\mathbf{2^{29.2}}$	$\mathbf{2^{64}}$

In Table 2 and 3, we let $n \leq 12$ in our algorithm, other parameters are explicitly listed in the tables. Here we only give the parameters that result in *uniform* complexities, i.e., the trade-off between data/time/memory and pre-computation complexities is as balanced as possible. Other choices of parameters are also allowed, but they do not have the uniform property.

As in [3,25], we have implemented the case that $L = 40$ and $p = 0.531$ in Table 2 in C on the platform mentioned above. It takes less than 4 seconds to restore the initial state of the involved LFSR after a pre-computation with negligible time. Compared to the attack in [3] which takes a few *days* for pre-computation and that in [25] whose pre-computation lasts for a few *hours*, the gain on efficiency in the preprocessing phase is obvious. This mainly comes from the fact that our algorithm can decode with much lower data complexities. In fact, the keystream requirement in our attack is 2 times smaller than that in [3] and is 2^6 times smaller than that in [25] in this implemented case. Other complexities listed in Table 2 and 3 are theoretical predictions derived according to (7) – (11) and Theorem 1.

From the above simulation results and theoretical estimates, we can see that our algorithm makes real-life fast correlation attacks much more reachable for the noise cases that only theoretical estimates are known by previous methods. This will have an impact on the choice of the secure parameters for the corresponding stream ciphers, as shown in Section 5.

5 Application to the Shrinking Generator

The shrinking generator (SG) was proposed in [4] at Crypto'93. So far, various key recovery and distinguishing attacks on the shrinking generator have been proposed [4,5,8,13,18,23,24], but none of them can practically threaten the security of the shrinking generator with the suggested parameters in [11]. In the following, we will demonstrate a key recovery attack against the same shrinking generator that is near-practical when measured in time/data/memory and preprocessing complexities.

More precisely, let the output sequence of data LFSR B used in a shrinking generator be $b = b_0, b_1, \cdots$. The cryptanalysis of the shrinking generator is usually composed of two phases. First, a new sequence $\hat{b} = \hat{b}_0, \hat{b}_1, \cdots$ associated with b by the relation $P(\hat{b}_i = b_i) = \frac{1}{2} + \varepsilon_i$ $(\varepsilon_i > 0)$ is constructed either by the method in [7] or by the method in [24]. Second, a decoding algorithm is applied to recover the initial state of LFSR B from $\hat{b}$. Here we do not focus on how the sequence $\hat{b}$ is constructed, but on how to efficiently exploit the existing correlations between b and $\hat{b}$. According to [24], the average biases between $\hat{b}$ and b for different keystream lengths N are shown in Table 4.

Table 4. The average biases for different keystream lengths N using the method in [24], which are obtained by a pre-computation of about 4 hours by Mathematica on a Pentium 4 processor

N	240	3000	8000	10000	140000
ε	0.0542	0.021	0.020	0.0195281	0.00982376

For the shrinking generator with the data LFSR B of length 61 and the control LFSR S of similar length, as suggested in [11], we show how to decode $\hat{b}$ using our algorithm in $2^{35.86}$ operations. From Table 4, the correlation between $\hat{b}$ and b is 0.5195281 if $N = 10000$. Note that 10146 keystream bits are needed to get a sequence $\hat{b}$ of 10000 bits. We choose the following parameters in our algorithm: $k = 27$, $n = 12$, $t = 5$, $T = 8.6 \times 10^8$, then $\Omega(\mathbf{v}_{L-k}) = 48457895$, $P_{right} = 97.42\%$ and $P_{wrong} = 2^{-32.16}$. Thus the total time complexity is $12 \cdot (2^{27} \cdot 27 + 48457895 \cdot (27 + 5)) = 2^{35.86}$ operations, the memory complexity is at most $32 \cdot 2^{27} + 48457895 \cdot 12 \cdot (61 + 5 \cdot \lceil \log_2 10000 \rceil) = 2^{36.23}$-bit after a precomputation of $10000^3 = 2^{39.9}$ operations. We can see that the data and time complexities of our attack are all practical, while the memory and preprocessing complexities are near-practical. The comparisons of our attack with other known attacks on the same target shrinking generator are given in Table 5, which shows that our attack is the best known attack against the shrinking generator with the suggested parameters.

At Eurocrypt'2003, a distinguishing attack on the shrinking generator is proposed in [5]. The best result given in [5] is to distinguish a shrinking generator with the data LFSR B having a weight 4 polynomial of degree 10000 using

Table 5. Comparisons of different attacks on the shrinking generator with the suggested parameters in [11]

	[18]	[8]	[13]	[24]	ours
N	few	$2^{10.23}$	2^{42}	140000	**10146**
$time$	2^{80}	2^{77}	2^{42}	2^{57}	$\mathbf{2^{35.86}}$

2^{32} output bits. Note that an arbitrary weight feedback polynomial of degree r is known to have a weight 4 multiple of degree around $2^{r/3}$ and $10000 = 2^{13.2877} = 2^{r/3}$, the distinguishing attack in [5] is only applicable to arbitrary data LFSR's of length around 40. Please see the following example.

Example 1. Consider the polynomial $x^{40} + x^{38} + x^{35} + x^{32} + x^{28} + x^{26} + x^{22} + x^{20} + x^{17} + x^{16} + x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$, it can be easily checked that its weight 4 multiple is $x^{24275} + x^{6116} + x^{1752} + 1$. Note that the degree 24275 is higher than the degree 10000 given in [5]. □

To further show the advantages of our decoding algorithm, consider a shrinking generator with the data LFSR B of length 40. We launch a key recovery attack on such a shrinking generator with the following attack parameters: $N = 8119$, $t = 3$, $k = 20$, $n = 9$, $p = 0.52$ and $T = 10^6$. Thus, given $2^{26.5}$-bit memory, the time complexity is $2^{27.62}$ operations after a pre-computation of 2^{26} operations. We implemented our attack in C on the same platform as that in Section 4. The realtime decoding lasts for about 10 seconds to output the correct initial state of LFSR B after a pre-computation of negligible time. Since an efficient key recovery attack is usually believed to be stronger than an efficient distinguishing attack on the same cipher, we conclude that our attack is stronger than that in [5].

Remarks on the security of the shrinking generator. For a shrinking generator with the data LFSR having a known connection, our results show that we should use a LFSR of longer length than that recommended by Krawczyk in [11]. If a security level of 2^{80} is needed, the length of the data LFSR should be at least 128-bit and the control LFSR should be of similar length. This comes from the following attack scenario: $L = 128$, $N = 140000$, $p = 0.50982376$, $k = 71$, $t = 6$ and $n = 16$. The corresponding time, memory and preprocessing complexities are $2^{81.15}$ operations, 2^{76}-bit and $2^{59.4}$ operations. An alternative way for strengthening the security of a shrinking generator is to use an unknown connection for the data LFSR at the expense of more hardware complexity. This is originally suggested by the designers in [4], but all the known cryptanalysis results on the shrinking generator so far are achieved under the known connection assumption. Our result on the shrinking generator could be seen as an end of such a research routine if a shrinking generator with the suggested parameters in [11] is employed.

6 Conclusions

In this paper, we proposed an improved fast correlation attack based on the combination of Johansson and Jönsson's algorithm with techniques for substituting keystream and evaluating parity-checks. Both the simulations and theoretical estimates show that the new algorithm is more efficient than all the previously known fast correlation attacks in general. The importance of such an algorithm is that the secure parameters formerly proposed for the corresponding stream ciphers have to be re-evaluated by our decoding method, which is verified by our cryptanalytic result on the shrinking generator with the parameters recommended by Krawczyk in 1994. We believe that our method will be useful in the cryptanalysis of those LFSR-based stream ciphers that other attacks, e.g. algebraic attacks, cannot deal with efficiently.

Acknowledgements. We would like to thank the anonymous reviewers for very helpful comments. This paper is supported by the National Natural Science Foundation of China (Grant No. 60603018) and the National High Technology Research and Development 863 Programm of China (No. 2007AA01Z470).

References

1. 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)
2. 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)
3. 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)
4. Coppersmith, D., Krawczyk, H., Mansour, Y.: The shrinking generator. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 22–39. Springer, Heidelberg (1994)
5. Ekdahl, P., Johansson, T.: Predicting the shrinking generator with fixed connections. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 330–344. Springer, Heidelberg (2003)
6. Goldreich, O., Rubinfeld, R., Sudan, M.: Learning polynomials with queries: the highly noisy case. In: 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, pp. 294–303 (1995)
7. Golić, J.D.: Correlation analysis of the shrinking generator. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 440–457. Springer, Heidelberg (2001)
8. Golić, J.D., O'Connor, L.: Embedding and probabilistic correlation attacks on clock-controlled shift registers. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 230–243. Springer, Heidelberg (1995)
9. Golić, J.D.: Iterative optimum symbol-by-symbol decoding and fast correlation attack. IEEE Trans. Inform. Theory 47, 3040–3049 (2001)
10. Golić, J.D., Hawkes, P.: Vectorial appraoch to fast correlation attacks. Designs, Codes and Cryptography 35, 5–19 (2005)

11. Krawczyk, H.: The shrinking generator: some practical considerations. In: Anderson, R. (ed.) FSE 1993. LNCS, vol. 809, pp. 45–46. Springer, Heidelberg (1994)
12. Johansson, T., Jönsson, F.: Fast correlation attacks based on turbo code techniques. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 181–197. Springer, Heidelberg (1999)
13. 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)
14. Johansson, T., Jönsson, 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)
15. Johansson, T., Jönsson, F.: Fast correlation attacks through reconstruction of linear polynomials. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 300–315. Springer, Heidelberg (2000)
16. 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)
17. Meier, W., Staffelbach, O.: Fast correlation attacks on certain stream ciphers. Journal of Cryptology, 159–176 (1989)
18. Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC, Boca Raton (1996)
19. Mihaljević, M.J., Fossorier, M.P.C., Imai, H.: A low-complexity and high-performance algorithm for the fast correlation attack. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 196–212. Springer, Heidelberg (2001)
20. Mihaljević, M.J., Fossorier, M.P.C., Imai, H.: Fast correlation attack algorithm with listing decoding and an application. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 196–212. Springer, Heidelberg (2001)
21. Shannon, C.E.: A Mathematical theory of communication. Bell Syst. Tech., J. 27 (1948)
22. Siegenthaler, T.: Decrypting a class of stream ciphers using ciphertext only. IEEE Transactions on Computer C-34, 81–85 (1985)
23. Simpson, L.R., Golić, J.D., Dawson, E.: A probabilistic correlation attack on the shrinking generator. In: Boyd, C., Dawson, E. (eds.) ACISP 1998. LNCS, vol. 1438, pp. 147–158. Springer, Heidelberg (1998)
24. Zhang, B., Wu, H., Feng, D., Bao, F.: A fast correlation attack on the shrinking generator. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 72–86. Springer, Heidelberg (2005)
25. Zhang, B., Feng, D.: Multi-pass fast correlation attack on stream ciphers. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 234–248. Springer, Heidelberg (2007)