Fast Correlation Attacks: an Algorithmic Point of View

Philippe Chose, Antoine Joux, and Michel Mitton

DCSSI, 18 rue du Docteur Zamenhof
F-92131 Issy-les-Moulineaux cedex, France
Philippe.Chose@ens.fr, Antoine.Joux@m4x.org
michelmitton@compuserve.com

Abstract. In this paper, we present some major algorithmic improvements to fast correlation attacks. In previous articles about fast correlations, algorithmics never was the main topic. Instead, the authors of these articles were usually addressing theoretical issues in order to get better attacks. This viewpoint has produced a long sequence of increasingly successful attacks against stream ciphers, which share a main common point: the need to find and evaluate parity-checks for the underlying linear feedback shift register. In the present work, we deliberately take a different point of view and we focus on the search for efficient algorithms for finding and evaluating parity-checks. We show that the simple algorithmic techniques that are usually used to perform these steps can be replaced by algorithms with better asymptotic complexity using more advanced algorithmic techniques. In practice, these new algorithms yield large improvements on the efficiency of fast correlation attacks.

Keywords. Stream ciphers, fast correlation attacks, match-and-sort, algorithmics, parity-checks, linear feedback shift registers, cryptanalysis.

1 Introduction

Stream ciphers are a special class of encryption algorithms. They generally encrypt plaintext bits one at a time, contrary to block ciphers that use blocks of plaintext bits. A *synchronous stream cipher* is a stream cipher where the ciphertext is produced by bitwise adding the plaintext bits to a stream of bits called the keystream, which is independent of the plaintext and only depends on the secret key and on the initialization vector. These synchronous stream ciphers are the main target of fast correlation attacks.

The goal in stream cipher design is to produce a pseudo-random keystream sequence, pseudo-random meaning indistinguishable from a truly random sequence by polynomially bounded attackers. A large number of stream ciphers use Linear Feedback Shift Registers (LFSR) as building blocks, the initial state of these LFSRs being related to the secret key and to the initialization vector. In nonlinear combination generators, the keystream bits are then produced by combining the outputs of these LFSRs through a nonlinear boolean function (see

Fig. 1). Many variations exist where the LFSRs are multiplexed or irregularly clocked.

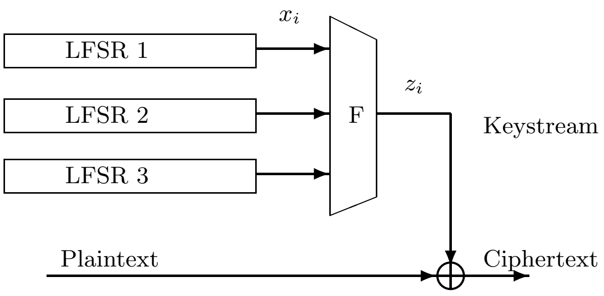

Fig. 1. Three LFSRs combined by a nonlinear boolean function

Among the different kinds of attacks against stream ciphers, correlation attacks are one of the most important [12, 13]. These cryptanalytic methods target nonlinear *combination* keystream generators. They require the existence of linear correlations between LFSR internal stages and the nonlinear function output, i.e. correlation between linear combinations of internal and output bits; these correlations need to be good enough for the attack to be successful. A very important fact about nonlinear functions is that linear correlations always exist [12]. Finding them can sometimes be the hardest part of the job, since the main method for finding them is by statistical experimentation. For the simplest ciphers, the correlations can be found by analyzing the nonlinear function with the well-known Walsh transform. Once a correlation is found, it can be written as a probability

$$p = \Pr(z_i = x_i^{j_1} \oplus x_i^{j_2} \oplus \ldots \oplus x_i^{j_M}) \neq 0.5$$

where z_i is the i-th keystream output bit and the $x_i^{j_1}, \ldots, x_i^{j_M}$ are the i-st output bits of some LFSRs $j_1, \ldots, j_M$ (see Fig. 1). The output can thus be considered as a noisy version of the corresponding linear combination of LFSR outputs. The quality of the correlation can be measured by the quantity $\varepsilon = |2p - 1|$. If ε is close to one, the correlation is a very good one and the cipher is not very strong. On the other hand, when ε is close to zero, the output is very noisy and correlation attacks will likely be inefficient. Since the LFSR output bits are produced by linear relations, we can always write this sum of output bits $x_i^{j_1} \oplus x_i^{j_2} \oplus \ldots \oplus x_i^{j_M}$ as the output of only one larger LFSR. Without loss of generality, the cipher can be presented as in Fig. 2, where this sum has been replaced by x_i the output of one LFSR, and the influence of the nonlinear function has been replaced by a BSC (binary symmetric channel), i.e. by a channel introducing noise with probability $1 - p$. Fast correlation attacks are an

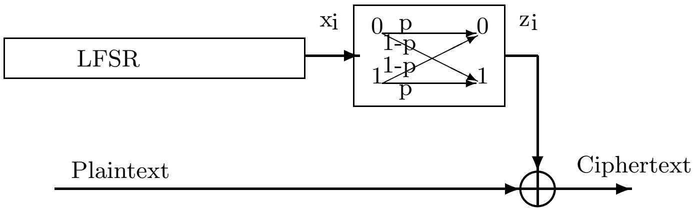

Fig. 2. Equivalent diagram where the output x_i of the LFSR is correlated with the keystream output z_i: $P(z_i = x_i) = p \neq 0.5$

improvement of the basic correlation attacks. They essentially reduce the time complexity of the cryptanalysis by pre-computing data [3, 5, 7, 8, 9].

In this article, we present substantial algorithmic improvements of existing fast correlation attacks. The paper is organized as follows. In Sect. 2, we introduce the basics of fast correlation attacks and a sketch of our algorithmic improvements. A detailed description of the algorithm is given in Sect. 3, together with its complexity analysis and, finally, some comparisons with other algorithms are provided in Sect. 4.

2 Fast Correlation Attacks

Fast correlation attacks are usually studied in the binary symmetric channel model as shown on Fig. 2. In this model, we consider the output of the generator as a noisy version of the output of some of the linear registers. The cryptanalysis then becomes a problem of decoding: given a noisy output, find the exact output of the registers and, when needed, reconstruct the initial filling of the registers.

The common point between all fast correlation attacks algorithms is the use of the so-called parity-check equations, i.e. linear relations between register output bits x_i. Once found, these relations can be evaluated on the noisy outputs z_i of the register. Since they hold for the exact outputs x_i, the evaluation procedure on the noisy z_i leaks information and helps to reconstruct the exact output sequence of the LFSR.

Fast correlation algorithms are further divided into iterative algorithms and one-pass algorithms. In iterative algorithms, starting from the output sequence z_i, the parity-checks are used to modify the value of these output bits in order to converge towards the output x_i of the LFSR thus removing the noise introduced by the BSC. The reconstruction of the internal state is then possible [2, 7]. In one-pass algorithms, the parity-checks values enable us to directly compute the correct value of a small number of LFSR outputs x_i from the output bits z_i of the generator. This small number should be larger than the size of the LFSR in order to allow full reconstruction [3, 5, 8, 9].

2.1 Sketch of One-Pass Fast Correlation Attacks

The one-pass correlation attack presented here is a variation of the attacks found in [3] and [9]. The main idea is, for each LFSR's output bit to be predicted (henceforth called *target bits*), to construct a set of estimators (the parity-check equations) involving k output bits (including the target bit), then to evaluate these estimators and finally to conduct a majority poll among them to recover the initial state of the LFSR.

This main idea is combined with a partial exhaustive search in order to yield an efficient cryptanalysis. More precisely, for a length–L LFSR, B bits of the initial state are guessed through exhaustive search and $L - B$ bits remain to be found using parity-checks techniques (see Fig. 3). However, for a given target bit, the result of the majority poll may lead to a near tie. In order to avoid this problem, we target more than $L-B$ bits, namely D and hope that at least $L-B$ will be correctly recovered.

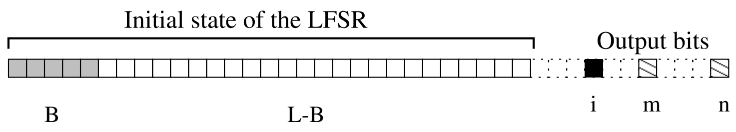

Fig. 3. Construction of parity-checks: In this example, the parity-check combines two bits of output (m and n) together with a linear combination of the B guessed bits in order to predict the target bit i

For each of these D target bits, we evaluate a large number Ω of estimators using the noisy z_i values and we count the number of parity-checks that are satisfied and unsatisfied, respectively N_s and $N_u = \Omega - N_s$. When the absolute value of the difference of these two numbers is smaller than some threshold θ, we forget this target bit. However when the difference is larger than the threshold, the majority poll is considered as successful. In that case, we predict $\hat{x}_i = z_i$ if $N_s > N_u$ and $\hat{x}_i = z_i \oplus 1$ otherwise (see Fig. 4). When the majority polls are successful and give the correct result for at least $L - B$ of the D target bits, we can recover the complete state of the LFSR using simple linear algebra.

2.2 New Algorithmic Ideas

When implementing the fast correlation attack from the previous section, several algorithmic issues arise. First we need to pre-compute the parity-checks for each target bit. Then we need to efficiently evaluate these parity-checks and recover the target bits. In previous papers, the latter step was performed using the straightforward approach, i.e. by evaluating parity-checks one by one for each possible guess of the first B bits, then by counting the number of positive and negative checks. For the preprocessing step, three algorithms were known,

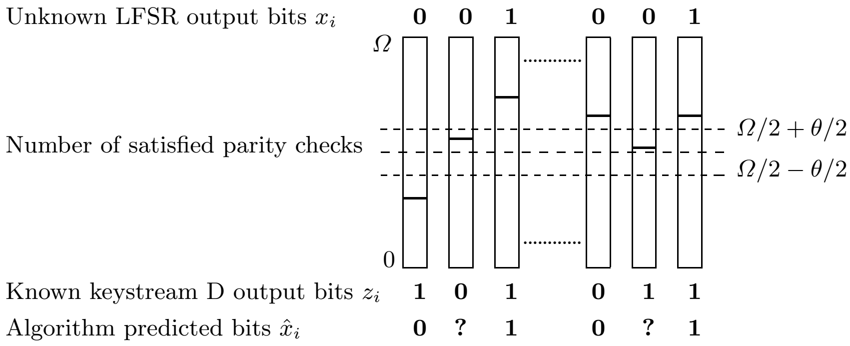

Fig. 4. Sketch of the decision procedure

simple exhaustive search, square-root time-memory tradeoffs and Zech's logarithm technique [10]. The main contribution of this paper is to propose better algorithmic techniques for both tasks.

Pre-processing Stage The usual square-root algorithm for computing parity-checks on k bits (the target bit plus $k-1$ output bits) works as follows. For each target bit, compute and store in a table the formal expression of the sum of $\lfloor \frac{k-1}{2} \rfloor$ output bits and the target bit in term of the initial L-bit state. Then sort this table. Finally compute the formal sum of $\lceil \frac{k-1}{2} \rceil$ output bits and search for a partial collision (on the $L-B$ initial bits, excluding the B guessed bits). The time and memory complexity of this algorithm are respectively $\mathcal{O}(DN^{\lceil (k-1)/2 \rceil} \log N)$ and $\mathcal{O}(N^{\lfloor (k-1)/2 \rfloor})$ where N is the length of the considered output sequence and D is the number of target bits. For even values of k, a different tradeoff exists: it yields a respective time and memory complexity $\mathcal{O}(N^{k/2} \log N)$ and $\mathcal{O}(N^{k/2})$. This square-root algorithm is part of a family of algorithms which can be used to solve a large class of problems. In some cases, there exists an alternative algorithm with the same time complexity as the original and a much lower memory complexity. A few examples are:

- the knapsack problem and modular knapsack problem [11, 1],
- the match and sort stage of SEA elliptic curve point counting [6],
- the permuted kernel problem [4].

Our goal is to propose such an alternative for constructing parity-checks. According to known results, we might expect a time complexity of $\mathcal{O}(\min(DN^{\lceil (k-1)/2 \rceil} \log N, N^{\lceil k/2 \rceil} \log N))$ and a memory complexity of $\mathcal{O}(N^{\lfloor k/4 \rfloor})$. It turns out that such an alternative really exists for $k \geq 4$, the algorithmics being given in Sect. 3.

Decoding Stage When using the usual method for evaluating the parity-checks, i.e. by evaluating every parity-check for each target bit and every choice of the B guessed bits, the time complexity is $\mathcal{O}(D2^B \Omega)$ where Ω is the number of such parity-check equations. By grouping together every parity-check involving the same dependence pattern on a well chosen subset of the B guessed bits (of size around $\log_2 \Omega$), it is possible to evaluate these grouped parity-checks in a single pass through the use of a Walsh transform. This is much faster than restarting the computation for every choice of the B bits. The expected time complexity of this stage then becomes $\mathcal{O}(D2^B \log_2 \Omega)$.

3 Algorithmic Details and Complexity Analysis

We present in this section a detailed version of our algorithmic improvements. We consider here that the parameters N, L, D, B, θ and ε are all fixed. The optimal choice of these parameters is a standard calculation and is given in appendix A.

Let us recall the notations used in the following:

- N is the number of available output bits;
- L is the length of the LFSR;
- B is the number of guessed bits;
- D is the number of target bits;
- x_i is the i-th output bit of the LFSR;
- z_i is the i-th output bit of the generator;
- $p = Pr(x_i = z_i) = \frac{1}{2}(1+\varepsilon)$ is the probability of correct prediction;

3.1 Pre-processing Stage

During the pre-processing stage, we search for all parity-checks of weight k associated with one of the D target bits. Following [9] and [2], we construct the set Ω_i of parity-check equations associated with the target bit i. This set contains equations of the form:

$$x_i = x_{m_1} \oplus \ldots \oplus x_{m_{k-1}} \oplus \sum_{j=0}^{B-1} c^j_{\boldsymbol{m},i} x_j$$

where the m_j are arbitrary indices among all the output bits and the $c^j_{\boldsymbol{m},i}$ are binary coefficients characterizing the parity-check. $\boldsymbol{m}$ stands for $[m_1, \ldots, m_{k-1}]$. In these equations, we express x_i as a combination of $k-1$ output bits plus some combination of the B guessed bits. The expected number of such parity-check equations for a given i is:

$$\Omega \approx 2^{B-L} \binom{N}{k-1}$$

For $k \leq 4$, the basic square-root time-memory tradeoff gives us these parity-checks with a time and memory complexity respectively of $\mathcal{O}(DN^{\lceil (k-1)/2 \rceil} \log N)$ and $\mathcal{O}(N^{\lfloor (k-1)/2 \rfloor})$.

In the sequel, we first solve a slightly more general problem. We try to find equations of the form:

$$A(\boldsymbol{x}) = x_{m_1} \oplus \ldots \oplus x_{m_{k'}} \oplus \sum_{j=0}^{B-1} c^j_{\boldsymbol{m},i} x_j$$

where $A(\boldsymbol{x}) = \sum_{j=0}^{L-1} a_j x_j$, $\boldsymbol{x} = [x_0, \ldots, x_{L-1}]$ and the a_j are fixed constants. When k is even, $A(\boldsymbol{x})$ will not be used and will be set to 0. When k is odd, $A(\boldsymbol{x})$ will be set to the formal expresion of one of the target bits and the problem will be similar to the even case. For the time being, let k' be the weight of the parity-check equation.

The main idea of the match-and-sort algorithm alternative we are going to use here is to split the huge task of finding collisions among $N^{k'}$ combinations into smaller tasks: finding less restrictive collisions on smaller subsets, sort the results and then aggregate these intermediate results to solve the complete task.

Algorithm 1 Find parity-checks of weight k' for a given $A(\boldsymbol{x})$

Evenly split k' between l_1, l_2, l_3 and l_4 with $l_1 \geq l_2$ and $l_3 \geq l_4$
for all choice of l_2 bits $(j_1 \ldots j_{l_2})$ **do**
  Formally compute $x_{j_1} \oplus \ldots \oplus x_{j_{l_2}} = \sum_{k=0}^{L-1} u_k x_k$
  Store in $U[\boldsymbol{u}] = \{j_1, \ldots, j_{l_2}\}$
end for
for all choice of l_4 bits $(m_1 \ldots m_{l_4})$ **do**
  Formally compute $x_{m_1} \oplus \ldots \oplus x_{m_{l_4}} = \sum_{k=0}^{L-1} v_k x_k$
  Store in $V[\boldsymbol{v}] = \{m_1, \ldots, m_{l_4}\}$
end for
for all $s = 0 \ldots 2^S - 1$ **do**
  for all choice of l_1 bits $(i_1 \ldots i_{l_1})$ **do**
    Formally compute $A(\boldsymbol{x}) \oplus x_{i_1} \oplus \ldots \oplus x_{i_{l_2}} = \sum_{k=0}^{L-1} c_k x_k$
    Search for $\boldsymbol{u}$ in U such that $\pi_S(\boldsymbol{u} \oplus \boldsymbol{c}) = s$
    Store in $C[\boldsymbol{u} \oplus \boldsymbol{c}] = \{i_1, \ldots, i_{l_1}, j_1, \ldots, j_{l_2}\}$
  end for
  for all choice of l_3 bits $(k_1 \ldots k_{l_3})$ **do**
    Formally compute $x_{k_1} \oplus \ldots \oplus x_{k_{l_3}} = \sum_{k=0}^{L-1} d_k x_k$
    Search for $\boldsymbol{v}$ in V such that $\pi_S(\boldsymbol{v} \oplus \boldsymbol{d}) = s$
    Let $\boldsymbol{t} = \boldsymbol{v} \oplus \boldsymbol{d}$
    Search for $\boldsymbol{c}$ in C such that $\pi_{L-B}(\boldsymbol{c} \oplus \boldsymbol{t}) = 0$
    Output $\{A(\boldsymbol{x}), i_{1\ldots l_1}, j_{1\ldots l_2}, k_{1\ldots l_3}, m_{1\ldots l_4}, \boldsymbol{c} \oplus \boldsymbol{t}\}$
  end for
end for

First evenly split k' between four integer l_1, l_2, l_3 and l_4 with $l_1 \geq l_2$ and $l_3 \geq l_4$, i.e. find l_1, l_2, l_3 and l_4 such that $l_1+l_2+l_3+l_4 = k'$ and for i from 1 to 4, $l_i = \lfloor \frac{k'}{4} \rfloor$ or $l_i = \lceil \frac{k'}{4} \rceil$. Compute the formal sums of l_2 output bits in terms of the initial L-bit state, $x_{j_1} \oplus \ldots \oplus x_{j_{l_2}} = \sum_{k=0}^{L-1} u_k x_k$. Let us write $\boldsymbol{u} = \{u_0, \ldots, u_{L-1}\}$.

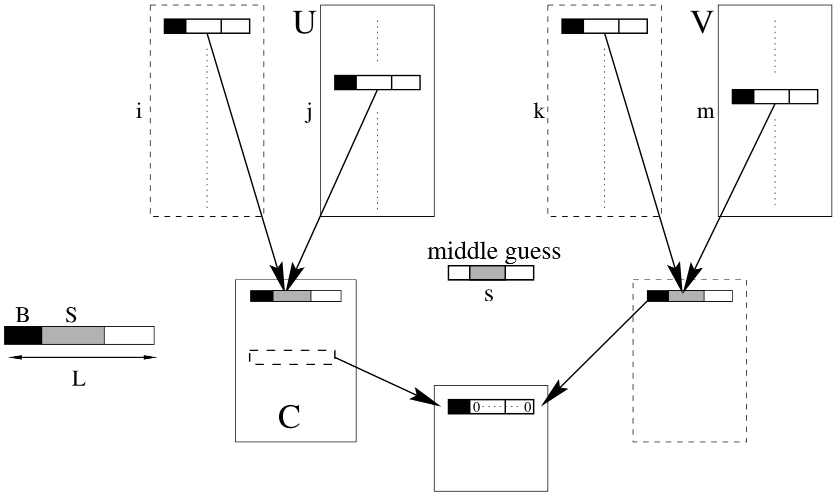

Fig. 5. Match-and-sort algorithm for finding parity-check equations

Store all these expressions in table U at entries $\boldsymbol{u}$. Do the same for a table V containing combinations of l_4 output bits. We will now try to match elements of table U with formal sums of l_1 output bits and elements of table V with formal sums of l_3 output bits (see Fig. 5). We only require the matching to be effective on a subset S of the $L - B$ bits. This S is chosen close to $\frac{k'}{4} \log_2 N$ in order to minimize the memory usage without increasing the time complexity. For each value s of the S bits, compute the formal sum $\boldsymbol{c}$ of $A(\boldsymbol{x})$ and of l_1 output bits in terms of initial bits and search for a partial collision in table U on S bits, i.e. find a $\boldsymbol{u}$ in U such that $\pi_S(\boldsymbol{u} \oplus \boldsymbol{c}) = s$ where π_S is the projection on the subspace spanned by the S bits. Store these collisions on S bits in a table C. Repeat the same procedure (finding partial collisions) with l_3 and l_4 replacing l_1 and l_2. For each found collision, search for a new collision combining the just found collision with an entry of table C, this time not on the S bits but on the full set of $L - B$ bits. Every found collision is a valid parity-check since it only involves the B guessed bits and $l_1 + l_2 + l_3 + l_4 = k'$ output bits (see Algorithm 1).

The time complexity of this algorithm is $\mathcal{O}(N^{\max(l_1+l_2, l_3+l_4)} \log N)$ and the memory complexity is $\mathcal{O}(N^{\max(\min(l_1,l_2),\min(l_3,l_4))})$, which for an even division of k' are equal respectively to $\mathcal{O}(N^{\lceil k'/2 \rceil} \log N)$ and $\mathcal{O}(N^{\lfloor (k'+1)/4 \rfloor})$. In fact, using the above algorithm for the homogenous case when $A(\boldsymbol{x}) = 0$ is a simple matter. Depending on the parity of k, two alternatives are possible. When k is odd, we simply let $k' = k - 1$ and let $A(\boldsymbol{x})$ represent x_i, one of the target bits. Of course we need to run the algorithm D times. When k is even, we let $k' = k$ and $A(\boldsymbol{x}) = 0$. In that case, we get the parity-checks for all output bits of the LFSR instead of merely D. With these choices, the complexities are listed in Table 1.

Table 1. Time and memory complexities of the new algorithm compared to those of the square-root algorithm

	New algorithm		Square-root algorithm (tradeoff 1)		Square-root algorithm (tradeoff 2)	
k	*Time*	*Memory*	*Time*	*Memory*	*Time*	*Memory*
4	$N^2 \log N$	N	$DN^2 \log N$	N	$N^2 \log N$	N^2
5	$DN^2 \log N$	N	$DN^2 \log N$	N^2	$DN^2 \log N$	N^2
6	$N^3 \log N$	N	$DN^3 \log N$	N^2	$N^3 \log N$	N^3
7	$DN^3 \log N$	N	$DN^3 \log N$	N^3	$DN^3 \log N$	N^3
8	$N^4 \log N$	N^2	$DN^4 \log N$	N^3	$N^4 \log N$	N^4
9	$DN^4 \log N$	N^2	$DN^4 \log N$	N^4	$DN^4 \log N$	N^4

3.2 Processing Stage

Decoding Part Let us write $B = B_1+B_2$ where B_1 and B_2 are positive integers to be determined. These two integers define two sets of bits in the initial state of the LFSR. Let us guess the B_1 bits of the initial state of the LFSR and denote by $\boldsymbol{X}_1$ the value of this guess. We regroup together all parity-check equations that involve the same pattern of the B_2 initial bits; let us rewrite each parity-check equation as:

$$z_i = \underbrace{z_{m_1} \oplus ... \oplus z_{m_{k-1}} \oplus \sum_{j=0}^{B_1-1} c^j_{\boldsymbol{m},i} x_j}_{t^1_{\boldsymbol{m},i}} \oplus \underbrace{\sum_{j=B_1}^{B_1+B_2-1} c^j_{\boldsymbol{m},i} x_j}_{t^2_{\boldsymbol{m},i}}$$

We group them in sets $M_i(\boldsymbol{c}_2) = \{\boldsymbol{m} \mid \forall j < B_2,\ c^{B_1+j}_{\boldsymbol{m},i} = c^j_2\}$ where $\boldsymbol{c}_2$ is a length B_2 vector and we define the function f_i as follows:

$$f_i(\boldsymbol{c}_2) = \sum_{\boldsymbol{m} \in M_i(\boldsymbol{c}_2)} (-1)^{t^1_{\boldsymbol{m},i}}$$

The Walsh transform of f_i is:

$$F_i(\boldsymbol{X}_2) = \sum_{\boldsymbol{c}_2} f_i(\boldsymbol{c}_2)(-1)^{\boldsymbol{c}_2 . \boldsymbol{X}_2}$$

When $\boldsymbol{X}_2 = [x_{B_1}, x_{B_1+1}, ..., x_{B-1}]$, we have $F_i(\boldsymbol{X}_2) = \sum_{\boldsymbol{m}} (-1)^{t^1_{\boldsymbol{m},i} \oplus t^2_{\boldsymbol{m},i}}$. So $F_i(\boldsymbol{X}_2)$ is the difference between the number of predicted 0 and the number of predicted 1 for the bit z_i, given the choice $\boldsymbol{X} = [\boldsymbol{X}_1, \boldsymbol{X}_2]$ for the B initial guessed bits. Thus a single Walsh transform can evaluate this difference for the 2^{B_2} choices of the B_2 bits.

The computation of $f_i(\boldsymbol{c}_2)$ for every $\boldsymbol{c}_2$ in $\mathbb{F}_2^{B_2}$ requires 2^{B_2} steps for the initialization and Ω steps for the evaluation of each parity-checks, whereas the Walsh transforms takes a time proportional to $2^{B_2} \log_2 2^{B_2} = 2^{B_2} B_2$. Since these calculations are done for every bit among the D considered ones and for each guess of the B_1 bits, the complexity of this part of the decoding is:

$$\begin{aligned} C_1 &= \mathcal{O}(2^{B_1} D(2^{B_2} + \Omega + 2^{B_2} B_2)) \\ &= \mathcal{O}(2^B D(\frac{\Omega}{2^{B_2}} + B_2)) \end{aligned}$$

Choosing $B_2 = \log_2 \Omega$, we get $C_1 = \mathcal{O}(2^B D \log_2 \Omega)$. This should be compared to the complexity using the straightforward approach: $C_1' = \mathcal{O}(2^B D\Omega)$.

Once $F_i(\boldsymbol{X}_2)$ is evaluated, predicting the corrected values $\hat{x}_i$ can be done with a simple procedure: for each i among the D considered bits, we have a function $F_i(\boldsymbol{X}_2)$. If the value of this function for a given value of $\boldsymbol{X}_2$ is far enough from zero, then the computed value of z_i that dominates among the $|\Omega_i|$ parity-check equations has a big probability to be the correct value of x_i (see Fig. 4). Let us call θ the threshold on the function $F_i(\boldsymbol{X}_2)$; we thus predict that, for a given $\boldsymbol{X}_2$:

$$x_i = \begin{cases} 0 \text{ if } F_i(\boldsymbol{X}_2) > \theta \\ 1 \text{ if } F_i(\boldsymbol{X}_2) < -\theta \end{cases}$$

Checking Part In order for our algorithm to be succesful, we need to have at least $L - B$ correctly predicted bits among the D considered bits. However, every predicted bit can sometimes be wrong. In order to increase the overall probability of success, let us introduce an extra step in the algorithm.

When the number of correctly predicted bits is less than $L - B$, the algorithm has failed. When this number is exactly equal to $L - B$, we can only hope that none of these bits is wrong. But when we have more than $L - B$ predicted bits, namely $L - B + \delta$, the probability that, among these bits, $L - B$ are correctly predicted greatly increase. So we add to the procedure an exhaustive search on all subset of size $L - B$ among the $L - B + \delta$ bits, in order to find at least one full correct prediction. Every candidate is then checked by iterating the LFSR and computing the correlation between the newly generated keystream x_i and the original one z_i.

If p_{err} is the probability that a wrong guess gives us at least $L - B + \delta$ predicted bits, then the checking part of the processing stage has a complexity of:

$$C_2 = \mathcal{O}((1 + p_{err}(2^B - 1)) \binom{L - B + \delta}{\delta} C_3)$$

since among the $2^B - 1$ wrong guesses, $p_{err}(2^B - 1)$ will be kept for checking and the 1 being there for the correct guess that should be kept. C_3 is the complexity of a single checking, i.e. $C_3 = \mathcal{O}(\frac{1}{\varepsilon^2})$.

The total complexity of the processing stage of the algorithm is then:

$$C = \mathcal{O}(2^B D \log_2 \Omega + (1 + p_{err}(2^B - 1)) \binom{L - B + \delta}{\delta} \frac{1}{\varepsilon^2})$$

4 Performance and Implementation

In this section, we present experimental and theoretical results of our algorithm applied to two LFSRs of lengths 40 and 89 bits. Optimal parameters were computed according to appendix A.

4.1 40-bit Test LFSR

Table 2. Complexity of the cryptanalysis for a probability of success p_{succ} close to 1. The LFSR polynomial is $1 + x + x^3 + x^5 + x^9 + x^{11} + x^{12} + x^{17} + x^{19} + x^{21} + x^{25} + x^{27} + x^{29} + x^{32} + x^{33} + x^{38} + x^{40}$.

Algorithm	*Noise*	*Required Sample*	*Complexity*
FSE'2001 [9]	0.469	400000	$\sim 2^{42}$
FSE'2001 [9]	0.490	360000	$\sim 2^{55}$
Our algorithm	0.469	80000	$\sim 2^{31}$
Our algorithm	0.490	80000	$\sim 2^{40}$

The chosen LFSR is the standard register used in many articles. The attack on the LFSR with noise $1-p = 0.469$ has been implemented in C on a Pentium III and provides results in a few days for the preprocessing stage and a few minutes for the decoding stage. After optimization of all the parameters ($D = 64$, $\delta = 3$, $B = 18$ and $k = 4$ for the first case, $D = 30$, $\delta = 3$, $B = 28$, $k = 4$ for the second one) the results are presented on Table 2. Results for $1 - p = 0.490$ are only theoretical. The gain on the complexity is at least equal to 2^{11}: it comes primarily from the Walsh transform. Moreover, the required length is five times smaller. This represents a major improvement on the time complexity of one-pass fast correlation attacks.

4.2 89-bit LFSR Theoretical Result

For a 89-bit LFSR, only theoretical results are provided on Table 3. The expected time complexity is 2^8 times smaller than previous estimations. Moreover the required sample length has decreased in a large amount. The parameters for our algorithm are $D = 128$, $\delta = 4$, $B = 32$ and $k = 4$.

Table 3. Complexity of the cryptanalysis for a probability of success p_{succ} close to 1 on a 89-bit LFSR

Algorithm	*Noise*	*Required Sample*	*Complexity*
FSE'2001 [9]	0.469	2^{38}	2^{52}
Our algorithm	0.469	2^{28}	2^{44}

5 Conclusion

In this paper, we presented new algorithmic improvements to fast correlation attacks. These improvements yield better asymptotic complexity than previous techniques for finding and evaluating parity-checks, enabling us to cryptanalyze larger registers with smaller correlations. Experimental results clearly show the gain on efficiency that these new algorithmic techniques bring to fast correlation attacks.

References

[1] D. Boneh, A. Joux, and P. Nguyen. Why textbook ElGamal and RSA encryption are insecure. In *Proceedings of ASIACRYPT'2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 2000.

[2] A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *Advances in Cryptology — EUROCRYPT'00*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. Springer Verlag, 2000.

[3] V. V. Chepyzhov, T. Johansson, and B. Smeets. A simple algorithm for fast correlation attacks on stream ciphers. In *Fast Software Encryption — FSE'00*, volume 1978 of *Lecture Notes in Computer Science*. Springer Verlag, 2000.

[4] É. Jaulmes and A. Joux. Cryptanalysis of pkp: a new approach. In *Public Key Cryptography 2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 165–172. Springer, 2001.

[5] T. Johansson and F. Jönsson. Fast correlation attacks through reconstruction of linear polynomials. In *Advances in Cryptology — CRYPTO'00*, volume 1880 of *Lecture Notes in Computer Science*, pages 300–315. Springer Verlag, 2000.

[6] A. Joux and R. Lercier. "Chinese & Match", an alternative to atkin's "Match and Sort" method used in the SEA algorithm. Accepted for publication in Math. Comp., 1999.

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

[8] M. Mihaljević, M. P. C. Fossorier, and H. Imai. A low-complexity and high-performance algorithm for fast correlation attack. In *Fast Software Encryption — FSE'00*, pages 196–212. Springer Verlag, 2000.

[9] M. Mihaljević, M. P. C. Fossorier, and H. Imai. Fast correlation attack algorithm with list decoding and an application. In *Fast Software Encryption — FSE'01*, pages 208–222. Springer Verlag, 2001. Pre-proceedings, final proceedings to appear in LNCS.

[10] W. T. Penzhorn and G. J. Kuhn. Computation of low-weight parity checks for correlation attacks on stream ciphers. In *Cryptography and Coding – 5th IMA Conference*, volume 1025 of *Lecture Notes in Computer Science*, pages 74–83. Springer, 1995.
[11] R. Schroeppel and A. Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981.
[12] T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Trans. on Information Theory*, IT-30:776–780, 1984.
[13] T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Trans. Comput.*, C-34:81–85, 1985.

A Optimal Parameters

In this appendix, we evaluate quantities needed in order to optimize parameters of the algorithm (B, D, θ and δ). We first look at the probability of successful decoding, i.e. the probability that the right guess gives us the right complete initial filling of the LFSR. Then we will evaluate the probability of false alarm, i.e. the probability that, having done a wrong guess, the algorithm outputs a full initial filling of the LFSR. This probability of false alarm enters in the complexity evaluation of our algorithm. In all our experimental results, the parameters were tuned to get a probability of success higher than 0.99.

A.1 Probability of Successful Decoding

Let us first suppose we have done the right guess for the B bits. Let us write $q = \frac{1}{2}(1+\varepsilon^{k-1})$ the probability for one parity-check equation to yield the correct prediction. Then the probability that at least $\Omega-t$ parity-check equations predict the correct result is:

$$P_1(t) = \sum_{j=\Omega-t}^{\Omega} (1-q)^{\Omega-j} q^j \binom{\Omega}{j}$$

Let t be the smallest integer such that $D\ P_1(t) \geq L - B + \delta$ (t is related to the former parameter θ by $\theta = \Omega - 2t$). Then we have, statistically, at least $L-B+\delta$ predicted bits in the selection of D bits, and we are able to reconstruct the initial state of the LFSR. The probability that at least $\Omega - t$ parity-check equations predict the wrong result is:

$$P_2(t) = \sum_{j=\Omega-t}^{\Omega} q^{\Omega-j} (1-q)^j \binom{\Omega}{j}$$

Let p_V be the probability that a bit is correctly predicted, knowing that we have at least $\Omega - t$ parity-check equations that predict the same value for this bit: $p_V = \frac{P_1(t)}{P_1(t)+P_2(t)}$. Then

$$p_{succ} = \sum_{j=0}^{\delta} \binom{L-B+\delta}{j} p_V^{L-B+\delta-j} (1-p_V)^j$$

is the probability that at most δ bits are wrong among the $L - B + \delta$ predicted bits, i.e. the probability of success of the first part of our algorithm.

A.2 Probability of False Alarm

The probability that a wrong guess gives at least $\Omega - t$ identical predictions among the Ω parity-check equations for a given bit i is

$$E(t) = \frac{1}{2^{\Omega-1}} \sum_{j=\Omega-t}^{\Omega} \binom{\Omega}{j}$$

since the probability that a parity-check equation is verified in this case is $\frac{1}{2}$. We can then deduce the probability that, with a wrong guess, more than $L - B + \delta$ bits are predicted, i.e. the probability of false alarm of the first part of our algorithm:

$$p_{err} = \sum_{j=L-B+\delta}^{D} \binom{D}{j} E(t)^j (1 - E(t))^{D-j}$$