An Improved Way to Construct the Parity-check Equations in Fast Correlation Attacks

Shiwei Chen
Zhengzhou Information Science and Technology Institute, Zhengzhou, Henan 450002, China
Science and Technology on Information Assurance Laboratory, Beijing 100072, China
Email: chensiwei1012@sohu.com

Chenhui Jin
Zhengzhou Information Science and Technology Institute, Zhengzhou, Henan 450002, China
Name of Institution/Department, City, Country
Email: jinchenhui@126.com

Abstract—How to construct the parity-check equations is an open problem in the topic of fast correlation attack. In this paper, we present an improved method to construct the parity-check equations used in the fast correlation attacks. By utilizing the idea of multi-layer match-and-sort combined with the exhaustive searching, we construct the parity-check equations to be used in the decoding, which is not relevant to the number of the LFSR’s feedback taps. Finally, we analyze the time complexity and memory complexity of our method, which are about $O(N^{(2^k/3)\log N})$ and $O(N^{(2^k-1)/3})$ respectively (where $k$ is the weight of the keystreams, such as the LFSR’s feedback taps), and the memory complexity is the square root of the Chose et al.’s.

Index Terms—Stream Cipher; Parity-Check Equations; LFSR; Complexity

I. INTRODUCTION

The stream cipher is one important kind of symmetric encryption algorithms, of which the goal is to produce a pseudo-random keystream sequence and then encrypt the plaintext bits by bitwise adding them to the keystream bits. To achieve the goal, the Linear Feedback Shift Registers (LFSRs) are used as the crucial building blocks, where the initial states are padded by secret keys or initialization vectors. However, due to the elements in the output sequence of the LFSR satisfy some simple linear equations in terms of the initial state, the initial state can be restored if we know enough long output sequence, which leads to that the LFSRs could not directly be used for the cryptographic applications. Therefore, in order to destroying the simple linear relations between the initial state and the output sequence, a common method is to use the LFSR’s outputs as the input of a suitably designed non-linear functions producing the keystreams, such as the nonlinear combination and the nonlinear filter keystream generators.

Among the different kinds of methods to analyze the security of the stream cipher based on the nonlinear combinations or nonlinear filter keystream generators, correlation attack is one of the most important algorithms [1]. The key step in the correlation attack is to find the correlations between linear combinations of the LFSR’s internal states and the output bits with the probability not equal to 0.5, which are called parity-check equations. Once the correlation is found, the output bit can be considered as a noisy of the corresponding linear combinations of LFSR’s internal states and thereby the initial state of the LFSR can be recovered by exhaustive searching. Then in [2], Meier and Staffelbach proposed the fast correlation attack, which essentially reduced the time complexity of the cryptanalysis by precomputations. Among all the fast correlation attacks, the parity-check equations describing the linear relations between LFSR’s output bits need to be constructed firstly. However, the construction of the parity-check equations is restricted by the number of LFSR’s feedback taps, such as the algorithms proposed in [3] and [4]. So, using the idea of match-and-sort, Chose et al. [5] accelerated the attacks in [3] and [4] by constructing the parity-check equations irrelevant to the number of the LFSR’s feedback taps. The main idea of Chose’s is combining exhaustive searching with the partial matching to yield an efficient cryptanalysis. By splitting the huge task of finding collisions among all the combinations into smaller tasks, and the memory complexity of their new algorithm was reduced and the time complexity was not changed compared to the square-root time-memory tradeoffs algorithm. In this paper, by splitting the huge task to find collisions into multi-layer smaller tasks, we present an improved way to construct the parity-check equations, of which the memory complexity is reduced to square root of the Chose’s.

The paper is organized as follows. In section 2, we introduce the ideas of the fast correlation attacks and some notations used through this article. In section 3, we describe the details of our improved method to construct the parity-check equations and analyze the complexity. Finally, we compare our results with the Chose et al.’s and conclude the paper.
II. DESCRIPTION OF THE FAST CORRELATION ATTACKS

A fast correlation attack was first proposed by Meier et al. [2] to analyze the security of the stream cipher based on LFSRs as the building blocks. So far, it has been applied successfully on some concrete stream ciphers, such as LILI-2 [6], Grain [7], et al. The fast correlation attacks mainly include two phases: in the first phase, a set of suitable parity-check equations are found and in the second phase, the initial state of the target LFSR is recovered by using these parity-check equations. Due to the use of the parity-check equations derived from the feedback polynomial of the LFSR, the fast correlation algorithms are significantly faster than exhaustive searching over the initial states of the LFSR. However, the construction of the parity-checks is affected by the number of feedback taps. If the number of the feedback taps is large, the fast correlation attack fails. So the stream ciphers usually utilize the LFSRs with many feedback taps to resist the fast correlation attack. However, by using the idea of match-and-sort combined with a partial exhaustive search, Chose et al. could construct a number of parity-checks in spite of the number of feedback taps.

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. 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.

For each of these D target bits, we evaluate a large number of estimators using the output keystream bits $z_i$ and the B guessed bits, and count the number of parity-checks satisfied $N_i$ and the number of parity-checks unsatisfied $N_u$. If the value $|N_i − N_u|$ is larger than a threshold, we predict $x_i = z_i$ if $N_i > N_u$ and $x_i = z_i \oplus 1$ otherwise. If the value $|N_i − N_u|$ is smaller than a threshold, we forget the target bit. When we obtain the correct result for at least (L − B) of the D target bits, we can recover the complete state of the LFSR by solving simple linear algebraic equations.

In this paper, we only focus on the construction of the parity-check equations. In follows, we describe the notations used through the paper.

- N: the number of the available output bits;
- L: the length of the LFSR;
- D: the number of target bits;
- B: the number of guessed bits;
- $x_i$: the i-th output bit of the LFSR;
- $z_i$: the i-th output bit of the generator;
- k: the weight of the parity-check equation.

III. OUR METHOD TO CONSTRUCT THE PARITY-CHECK EQUATIONS

In the fast correlation attacks, we need to construct all the parity-check equations of weight k associated with one of the D target bits firstly in the precomputation. Assume that $\Lambda_i$ is the set of parity-check equations associated with the i-th target bit $x_i$. This set contains the equations with the following form:

$$x_i = x_m \oplus x_m \oplus \cdots \oplus x_m \oplus \bigoplus_{j=0}^{B-1} c_{m,j} x_j$$

where $m_i (1 \leq j \leq k − 1)$ could be any indices among all the output bits and $c_{m,j}$ is binary coefficients characterizing the parity-check. Since the $x_i$ is expressed by the $k − 1$ output bits and the B guessed bits, the expected number of such equations is about $2^{B−L} \binom{N}{k−1}$

In [5], Chose et al. present an algorithm to search for parity-check equations of the following general form:

$$A(x) = x_m \oplus x_m \oplus \cdots \oplus x_m \oplus \bigoplus_{j=0}^{B-1} c_{m,j} x_j \quad (1)$$

where $A(x) = \sum_{j=0}^{L−1} a_j x_j$ and $a_j (0 \leq j \leq L−1)$ are fixed constants. When k is odd, let $k' = k − 1$ and $A(x) = x_i$. When k is even, let $k' = k$ and $A(x) = 0$. Throughout this paper, let $k'$ be the weight of the parity-check equation, that is, the number of the terms with non-zero coefficient.

The main idea of the Chose et al.’s algorithm is firstly splitting the $k'$ into four integers $l_1, l_2, l_3, l_4$ with $l_1 + l_2 + l_3 + l_4 = k', l_1 \geq l_2, l_3 \geq l_4$, and then utilized the ideas of match-and-sort to find all the solutions satisfying the above equation with the time complexity of $O(N^{|l_1, l_2, l_3, l_4|} \log N)$ and the memory of $O(N^{|l_1, l_2, l_3, l_4|})$

By studying the idea of match-and-sort used in Chose et al.’s algorithm, we find that splitting the huge task of finding collisions among $N^k$ combinations into smaller tasks could improve the complexity of constructing such parity-check equations. Hence, we develop a new technique called multi-layer match-and-sort to present an improved algorithm to constructing the parity-check equations used in the fast correlation attacks, of which the memory complexity is reduced to the square root of the Chose et al.’s.

A. Basic Theory of the Multi-Layer Match-And-Sort Technique

In the equation (1), the B bits are guessed, so $\sum_{j=0}^{B-1} c_{m,j} x_j$ is known. Denote $\sum_{j=0}^{B-1} c_{m,j} x_j = b$, where b is a constant, and we have

$$x_m \oplus x_m \oplus \cdots \oplus x_m = A(x) \oplus b$$

Hence, there exists a value s such that
of the all choices of the \(i\) output bits, and search for matches with the elements in \(U^2\) occurring on \(S_1\) bits. That is, for each value \(i\) of \(S_1\) bits, find a \(u^2\) in \(U^2\) such that \(\pi_{S_1}(u^2 \oplus u^2) = i\), where \(\pi_{S_1}\) is the projection on the subspace spanned by the \(S_1\) bits. Store the matches in table \(T_i\). Similarly, for \(i\), compute the formal sums of the all choices of the \(i\) output bits and search for matches with the elements in \(U^4\) on the value \(\pi_{S_1}(s) - i\) of \(S_1\) bits where \(s\) is of \(S\) bits. Meanwhile, do the same computations for \(i\) and \(i\), and store the matches in \(T^T\) and \(T^T\) respectively. On the second layer, for the first two sets and last two sets, we search for matches on the value \(s\) of \(S\) bits respectively. On the third layer, we require that the matching occurs on the last L-B-S bits. Followed it, we can obtain the collisions on the L-B bits.

See the Algorithm 1 for the details of our way to find the parity-checks of weight \(k'\) with \(k' \geq 8\).

**Algorithm 1** Find parity-check equations of weight \(k'\) with \(k' \geq 8\)

**Step 1** Split \(k'\) into \(l_1, l_2, \ldots, l_k\) with \(\sum_{i=1}^{8} l_i = k'\) and \(l_{j+1} \geq l_j (1 \leq j \leq 4)\)

**Step 2** For all \(j = 1, 2, 3, 4\) do

**Step 2.1** For all choice of \(l_j\) bits \((p_1^{(2j)}, p_2^{(2j)}, \ldots, p_{l_j}^{(2j)})\)

- Formally compute

\[
x_{p_1^{(2j)}} \oplus x_{p_2^{(2j)}} \oplus \cdots \oplus x_{p_{l_j}^{(2j)}} = \sum_{i=1}^{l_j} u_i^{(2j)} x_i
\]

- Store in

\[
U_{j}\{u_i^{(2j)}\} = \{p_1^{(2j)}, p_2^{(2j)}, \ldots, p_{l_j}^{(2j)}\}
\]

where \(u_i^{(2j)} = (u_0^{(2j)}, u_1^{(2j)}, \ldots, u_{l_j}^{(2j)})\)

**End for**

**End for**

**Step 3** For all choice of \(s = 0, 1, \ldots, 2^6 - 1\) do

**Step 3.1** For all choice of \(l_i\) bits \((p_1^{(i)}, p_2^{(i)}, \ldots, p_{l_i}^{(i)})\)

- Formally compute

\[
x_{p_1^{(i)}} \oplus x_{p_2^{(i)}} \oplus \cdots \oplus x_{p_{l_i}^{(i)}} = \sum_{i=1}^{l_i} u_i^{(i)} x_i
\]

- Search for \(u^2\) in the table \(U^2\) satisfying \(\pi_{S_1}(u^2 \oplus u^2) = i\)

Store in

\[
T_i\{u^2 \oplus u^2\} = \{p_1^{(i)}, p_2^{(i)}, \ldots, p_{l_i}^{(i)}, p_1^{(2i)}, p_2^{(2i)}, \ldots, p_{l_j}^{(2i)}\}
\]

**End for**

**Step 3.1.2** For all choice of \(l_3\) bits \((p_1^{(3)}, p_2^{(3)}, \ldots, p_{l_3}^{(3)})\)

- Formally compute

\[
x_{p_1^{(3)}} \oplus x_{p_2^{(3)}} \oplus \cdots \oplus x_{p_{l_3}^{(3)}} = \sum_{i=1}^{l_3} u_i^{(3)} x_i
\]

- Search for \(u^4\) in the table \(U^4\) satisfying

\[
\pi_{S_1}(u^4 \oplus u^4) = \pi_{S_2}(u^4 \oplus u^4) = \pi_{S_3}(u^4 \oplus u^4) = \pi_{S_4}(u^4 \oplus u^4)
\]
\[ \pi(s) (u^i \oplus u^3) = \pi(s) (s) - t_i \]

Let \( t^2 = u^4 \oplus u^3 \). Search for \( t^1 \) in table \( T^1 \) satisfying
\[ \pi(s) (t^1 \oplus t^3) = s \]

Store in
\[ W[t^1 \oplus t^2] = \{ p^{(j)}, 1 \leq j \leq 4, 1 \leq i \leq l_j \} \]

End for

End for

Step 3.2 For all choice of \( t_2 = 0, 1, \cdots, 2^5 - 1 \) do

Step 3.2.1 For all choice of \( l_5 \) bits
\( (p^{(5)}_1, p^{(5)}_2, \cdots, p^{(5)}_6) \) do

Formally compute
\[ A(x) \oplus x_{p^{(5)}_1} \oplus x_{p^{(5)}_2} \oplus \cdots \oplus x_{p^{(5)}_6} = \sum_{j=0}^{l_5-1} u^j x_j \]

Search for \( u^6 \) in the table \( U^6 \) satisfying
\[ \pi(s) (u^6 \oplus u^3) = t_2 \]

Store in
\[ T[U^6 \oplus u^3] = \{ p^{(5)}_1, p^{(5)}_2, \cdots, p^{(5)}_6, p^{(6)}, p^{(6)}, \cdots, p^{(6)} \} \]

End for

Step 3.2.2 For all choice of \( l_7 \) bits
\( (p^{(7)}_1, p^{(7)}_2, \cdots, p^{(7)}_6) \) do

Formally compute
\[ x_{p^{(7)}_1} \oplus x_{p^{(7)}_2} \oplus \cdots \oplus x_{p^{(7)}_6} = \sum_{j=1}^{l_7-1} u^j x_j \]

Search for \( u^8 \) in the table \( U^8 \) satisfying
\[ \pi(s) (u^8 \oplus u^7) = \pi(s) (s) - t_2 \]

Let \( t^4 = u^8 \oplus u^7 \).

Search for \( t^3 \) in table \( T^3 \) satisfying \( \pi(s) (t^3 \oplus t^4) = s \)

Let \( w^2 = t^3 \oplus t^4 \)

Search for \( w^1 \) in table \( W^1 \) satisfying
\[ \pi_{\text{L} \rightarrow \text{B}} (w^1 \oplus w^2) = 0 \]

Output \( \{ A(x), p^{(j)}_1, 1 \leq j \leq 4, 1 \leq i \leq l_j \} \)

End for

End for

C. Analysis of the Complexity of Our New Method

Now we analyze the complexity of our method to construct the parity-check equations in follows.

In Step 2, for each \( l_j \), with \( 1 \leq j \leq 4 \), we need to compute \( N^{l_j} \) formal sums and store the \( N^{l_j} \) formal sums. Then in Step 3, for every \( l_{j+1} \), with \( 1 \leq j \leq 4 \), we firstly need to compute \( N^{l_{j+1}} \) formal sums and then use the quick sort to search for the matches with the elements in table \( U^2 \) with the time complexity of about \( N^{l_{j+1}} \log N^{l_{j+1}} \).

Meanwhile, for \( l_1 \) and \( l_2 \), we store all the matching pairs in table \( T^1 \) and \( T^3 \) respectively. Since the number of elements in \( U^2 \) is \( N^{l_{j+1}} \) and there are \( N^{l_{j+1}} \) choices for the \( l_{j+1} \) bits \( (p^{(j+1)}_1, p^{(j+1)}_2, \cdots, p^{(j+1)}_{l_{j+1}}) \), we store \( N^{l_{j+1}} N^{l_{j+1}} \) pairs in table \( T^1 \) and \( N^{l_{j+1}} N^{l_{j+1}} \) pairs in table \( T^3 \), which only match on the \( S_i \) bits for each \( t_i (0 \leq t_i \leq 2^5 - 1) \) and \( t_j (0 \leq t_j \leq 2^5 - 1) \) respectively.

For \( l_1 \), we search for \( t^1 \) in table \( T^1 \) satisfying \( \pi(s) (t^1 \oplus t^3) = s \) and store the matching pairs in \( W^1 \), so the time complexity is about \( N^{l_{j+1}} N^{l_{j+1}} \log N^{l_{j+1}} \) and memory complexity is about \( N^{l_{j+1}} N^{l_{j+1}} N^{l_{j+1}} N^{l_{j+1}} \log N^{l_{j+1}} 2^{l_{j+1}} \). For \( l_1 \), search for \( t^1 \) in table \( T^3 \) satisfying \( \pi(s) (t^1 \oplus t^4) = s \) and then search for \( w^1 \) in table \( W^1 \) satisfying \( \pi(s) (w^1 \oplus w^2) = s \), so the time complexity is about \( N^{l_{j+1}} N^{l_{j+1}} \log N^{l_{j+1}} 2^{l_{j+1}} \)

\[ \log N^{l_{j+1}} N^{l_{j+1}} N^{l_{j+1}} N^{l_{j+1}} 2^{l_{j+1}} 2^{l_{j+1}} \]

In total, the time complexity is about \( O(\max(2^{l_{j+1}} N^{l_{j+1}} N^{l_{j+1}} N^{l_{j+1}} N^{l_{j+1}} 2^{l_{j+1}} N^{l_{j+1}} \log N)) \)

And the memory is about
\[ N^{l_{j+1}} + N^{l_{j+1}} + N^{l_{j+1}} + N^{l_{j+1}} \]

Specifically, choose \( S = \frac{k'}{4} \log N, S_i = \frac{k'}{8} \log N \), and we have the time complexity is about \( O(\sqrt{4^{l_{j+1}} \log N}) \) and the memory is about \( O(N^{l_{j+1} (4^{l_{j+1}} \log N)}) \), of which the memory is square root of the Chose et al.'s.

Remark 1 In the description of our method, we assume that the weight of the parity-check equations \( k' \geq 8 \). In fact, our method also applied to the case that the weight \( k' < 8 \). For example, if \( k' = 6 \), then we could repeat at least two of the nonzero terms to assure the weight is 8 and perform our method as described in Section B.
From Section III, we know that when \( k \) is odd, \( k' = k - 1 \) and \( A(x) = x_1 \), and when \( k \) is even, \( k' = k \) and \( A(x) = 0 \).

With different choices of \( k \), we compare our complexity with the Chose et al.’s in Table 1.

IV. CONCLUSIONS

In this paper, an improved method to construct the parity-check equations used in the fast correlation attacks is presented. By utilizing the idea of multi-layer match-and-sort combined with the exhaustive search, we construct the parity-check equations to be used in the decoding, which is not relevant to the number of the LFSR’s feedback taps. Finally, we analyze the time complexity and memory of our method, where the memory is square root of the Chose et al.’s. Furthermore, our method can be used to solve the knapsack problem, et al.

REFERENCES


Chenhui Jin, Fugou, Henan, China, 1965, Professor, mainly majored in cryptography.