Rewriting Variables: the Complexity of Fast Algebraic Attacks on Stream Ciphers

Philip Hawkes and Gregory G. Rose
Qualcomm International, Australia
Level 3, 230 Victoria Road, Gladesville, NSW 2111 Australia
{phawkes,ggr}@qualcomm.com

Abstract

Recently proposed algebraic attacks [AK03,CM03] and fast algebraic attacks [A04,C03] have provided the best analyses against some deployed LFSR-based ciphers. The process complexity is exponential in the degree of the equations. Fast algebraic attacks were introduced [C03] as a way of reducing run-time complexity by reducing the degree of the system of equations. Previous reports on fast algebraic attacks [A04,C03] have underestimated the complexity of substituting the keystream into the system of equations, which in some cases dominates the attack. We also show how the Fast Fourier Transform (FFT) [CT65] can be applied to decrease the complexity of the substitution step, although in one case this step still dominates the attack complexity. Finally, we show that all functions of degree *d* satisfy a common, function-independent linear combination that may be used in the pre-computation step of the fast algebraic attack and provide an explicit factorization of the corresponding characteristic polynomial. This yields the fastest known method for performing the pre-computation step.

1 Introduction

Many popular stream ciphers are based on linear feedback shift registers (LFSRs) [R91]. Such ciphers include E0 [B01], LILI-128 [SDGM00] and Toyocrypt [MI02]. They consist of a memory register called the *state* that is updated (changed) every time a keystream output is produced, and an additional device, called the nonlinear combiner. The nonlinear combiner computes a keystream output as a function of the current LFSR state.[1] The sequence of states produced by an LFSR depends on the initial state of LFSR, which is always presumed to be secret. Since recovering this initial state allows prediction of unknown keystream, we follow the convention of [C03] and call it $\boldsymbol{K}$ as if it was actually the key. Most practical stream ciphers initialize this state from the real key and a nonce. The advantages of LFSRs are many. LFSRs can be constructed very efficiently in hardware and some recent designs are also very efficient in software. LFSRs can be chosen such that the produced sequence has a high period and good statistical properties.

While there are many approaches to the cryptanalysis of LFSR-based stream ciphers, this paper is concerned primarily with the recently proposed algebraic attacks [AK03,CM03] and fast algebraic attacks [A04,C03]. Such attacks have provided the best analyses against some theoretical and deployed ciphers.

[1] Some LFSR-based stream ciphers have a non-linear filter that maintains some bits of memory, but research has shown that such ciphers can be analyzed in the same way as ciphers without memory. Some designs use multiple LFSRs, but again these are usually equivalent to a single LFSR. Some modern stream ciphers use units larger than bits, but this discussion applies equally to such ciphers, so we will talk only in terms of bits.

An algebraic attack consists of three steps. The first step is to find a system of algebraic equations that relate the bits of the initial state $\boldsymbol{K}$ and bits of the keystream $\boldsymbol{Z} = \{z_t\}$. Some methods [AK03,CM03] have been proposed for finding "localized" equations (where the keystream bits are in a small range $z_t,\ldots,z_{t+\theta}$). This first step is a pre-computation: the attacker must compute these equations before attacking a key-stream. Furthermore, the computation need only be performed once, and the attacker can use the same equations for attacking multiple key-streams. The second and third steps are performed after the attacker has observed some keystream. In the second step, the observed keystream bits are substituted into the algebraic equations (from the first step) to obtain a system of algebraic equations in the bits of $\boldsymbol{K}$. The third step is to solve these algebraic equations to determine $\boldsymbol{K}$. This will be possible if the equations are of low degree in the bits of $\boldsymbol{K}$, and a sufficient number of equations can be obtained from the observed keystream.

The process complexity of the third step is exponential in the degree of the equations. Fast algebraic attacks were introduced by Courtois at Crypto 2003 [C03] as a way of reducing run-time complexity by reducing the degree of the system of equations. This method requires an additional pre-computation step; this step determines a linear combination of equations in the initial system that cancels out terms of high degree (provided the algebraic equations are of a special form). This yields a second system of equations relating $\boldsymbol{K}$ and the keystream $\boldsymbol{Z}$ that contains only terms of low degree. In the second step, the appropriate keystream values are now substituted into this second system to obtain a new system of algebraic equations in the bits of $\boldsymbol{K}$. Solving the new system (in the third step) is easier than solving the old system because the new system contains only terms of low degree.

Courtois [C03] proposes using a method based on the Berlekamp-Massey algorithm [M69] for determining the linear combination obtained in the additional pre-computation step. The normal Berlekamp-Massey algorithm has a complexity of D^2, while an asymptotically-fast implementation has a complexity of $C \cdot D$ (log D) for some large constant C. It is unclear which method would be best for the size of D considered in these attacks. Armknecht [A04] provides a method for improving the complexity when the cipher consists of multiple LFSRs.

Contributions of this paper. The first contribution is to note that previous reports on fast algebraic attacks (such as [A04,C03]) appear to have underestimated the complexity of substituting the keystream into the second system of equations.[2] The complexity was originally underestimated as only $O(DE)$ [C03], where D is the size of the linear combination and E is the size of the second system of equations. Table 1 lists the values of $O(DE)$ for previously published attacks from [A04,C03]. However, simple substitution would require a complexity of $O(DE^2)$ (see Section 2.3), and no other method was suggested for reducing the complexity. It is true that E bitwise operations of the substitution can be performed in parallel, reducing the *time* complexity to $O(DE)$, but in cases where E is large, the *process* complexity should still be considered $O(DE^2)$ in the

[2] We are aware (via private communication) of other proposed algebraic attacks in which the substitution complexities were initially ignored. In one case, the complexity of simple substitution was almost the square of the complexity of solving the system!

absence of specialized hardware. In many cases DE^2 actually exceeds the complexity of solving the system of equations, as shown in Table 1.

The second contribution of this paper is to show how the Fast Fourier Transform (FFT) [CT65] can be applied to decrease the complexity of the substitution step from the naïve $DE^2/2$ to $2ED\log_2 D$. The resulting complexities of the FFT approach are also listed in Table 1.

Cipher	D	E	Data req	Substitution			Solving System	Total
				Claimed (wrong)	*Simple*	*FFT*		
E0	2^{23}	2^{18}	2^{24}	2^{41}	2^{59}	$\mathbf{2^{47}}$	2^{49}	2^{49}
LILI-128	2^{21}	2^{12}	2^{60}	2^{33}	2^{44}	$\mathbf{2^{39}}$	2^{39}	$\mathbf{2^{40}}$
Toyocrypt	2^{18}	2^{7}	2^{18}	2^{23}	2^{31}	$\mathbf{2^{30}}$	2^{20}	$\mathbf{2^{30}}$

Table 1. Comparison of the substitution complexities for the published fast algebraic attacks.

The final contribution of this paper is to provide an efficient method for determining the linear combination obtained in the additional pre-computation step of the fast algebraic attack. First we show that all functions of degree d satisfy a common function-independent linear combination of length $D = \sum_{i=0..d} C_n^d$ that is defined exclusively by the LFSR. Then we provide a direct method for computing this linear combination (based on the work of Key [K76]). This method requires $c \cdot D (\log D)^2$ operations, where c is a small constant. This is a significant improvement on the complexities of previous methods.

Cipher	D	Courtois [C03]: based on Berlekamp-Massey		Armknecht [A04]	**Direct Computation (this paper)**
		$C \cdot D (\log D)$	D^2	Parallel Method	$\boldsymbol{D (\log D)^2}$
E0	2^{23}	$C \cdot 2^{28}$	2^{46}	2^{43}	$\mathbf{2^{32}}$
LILI-128	2^{21}	$C \cdot 2^{26}$	2^{42}	-	$\mathbf{2^{30}}$
Toyocrypt	2^{18}	$C \cdot 2^{23}$	2^{36}	-	$\mathbf{2^{26}}$

Table 2. The complexities of the pre-computation step for published attacks, where C represents a large constant.

This paper is organized as follows: in Section 2 we describe fast algebraic attacks. In Section 3 we discuss the complexity of substitution step for fast algebraic attacks. Section 4 reviews some useful properties of FFTs. Section 5 describes how the Fast Fourier Transform can be applied to speed up the substitution step. Section 6 contains our observations about the linear combination used in the fast algebraic attack. Section 7 concludes the paper.

2 Fast Algebraic Attacks

The length of the LFSR is n-bits; that is the internal state of the LFSR is $\boldsymbol{K}_t \in \mathrm{GF}(2)^n$. A state $\boldsymbol{K}_{t+1}$ is derived from the previous state $\boldsymbol{K}_t$ by applying an (invertible) linear mapping $L : \mathrm{GF}(2)^n \rightarrow \mathrm{GF}(2)^n$, with $\boldsymbol{K}_{t+1} = L(\boldsymbol{K}_t)$. The function L can be represented by an $n \times n$

matrix over $GF(2)$, which is called the *state update matrix*. Notice that we can write $\boldsymbol{K}_t = L_t(\boldsymbol{K})$. Each keystream bit is generated by first updating the LFSR state (by applying L) and then applying a Boolean function to the bits of the LFSR state. For the purposes of this paper, everything about the cipher is presumed to be known to the attacker, except the initial state of the LFSR and any subsequent state derived from it.

Linearization: Recall that the first two steps of the attack result in a system of nonlinear algebraic equations in a small number of unknown variables (these variables being the bits of the initial state). The most successful algebraic attacks (to date), have been based on linearization. The basis of this technique is to "linearize" a system of nonlinear algebraic equations by assigning a new unknown variable to each monomial term that appears in the system. The same monomial term appearing in distinct equations is assigned the same new unknown variable. The system of equations then changes from a system of non-linear equations (with few unknown variables) into a system of linear equations (with a large number of unknown variables). If the number of linear equations exceeds the number of new unknown variables, then an attacker can solve the system to obtain the new unknown variables of the linear system (which will in turn reveal the unknown variables of the non-linear system). The advantage of linearization is that the attacker can use the large body of knowledge about the solution of linear systems.

2.1 The Monomial State

This section introduces some notation that is useful for describing linearization. For a given value of the state $\boldsymbol{K}_t$ and for a given degree d, we shall let $\boldsymbol{M}_d(t)$ *(the monomial state)* denote the $GF(2)$ column vector with each component being a corresponding monomial of degree d or less. The number of such monomials is $D = \sum_{i=0..d} C^n_i \sim C^n_d$, so $\boldsymbol{M}_d(t)$ contains D components. The initial monomial state $\boldsymbol{M}_d$ corresponds to the initiate state $\boldsymbol{K}$.

Example 1. If $n=4$ (that is, $\boldsymbol{K}_t = k_3\, k_2\, k_1\, k_0$) and $d=2$, then there are $D=11$ monomials of degree ≤ 2:

$$\begin{aligned}\boldsymbol{M}_d(t) &= (m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10})^{\mathrm{T}} \\ &= (1,\ k_0, k_1, k_2\ , k_3, k_0\,k_1, k_0\,k_2, k_0\,k_3, k_1\,k_2, k_1\,k_3, k_2\,k_3)^{\mathrm{T}}\end{aligned}$$

where the letter "T" denotes the transpose of the matrix to make a column vector. If $\boldsymbol{K}_t = 0111$, then the values of the monomials are:

$$\begin{aligned}\boldsymbol{M}_d(t) &= (1, k_0{=}1, k_1{=}1, k_2{=}1, k_3{=}0, k_0\,k_1{=}1, k_0\,k_2{=}1, k_0\,k_3{=}0, k_1\,k_2{=}1, k_1\,k_3{=}0, k_2\,k_3{=}0)^{\mathrm{T}} \\ &= (1,1,1,1,0,1,1,0,1,0,0)^{\mathrm{T}}\end{aligned}$$

Note that the ordering of the monomials is arbitrary; for consistency we will enumerate using lower subscripts first, as shown. ■

We can express a Boolean function of the state in terms of the monomial state.

Example 2. Consider a Boolean function of the state such as the function $f(\mathbf{K}) = k_2 + k_1 k_3$ (using the LFSR state from Example 1). This function can be expressed as

$$\begin{aligned} f(\mathbf{K}_t) &= k_2 + k_1 k_3 \\ &= \underline{0}\times 1 + \underline{0}\times k_0 + \underline{0}\times k_1 + \underline{1}\times k_2 + \underline{0}\times k_3 + \underline{0}\times k_0 k_1 \\ &\quad + \underline{0}\times k_0 k_2 + \underline{0}\times k_0 k_3 + \underline{0}\times k_1 k_2 + \underline{1}\times k_1 k_3 + \underline{0}\times k_2 k_3 \\ &= (0,0,0,1,0,0,0,0,0,1,0)\cdot \mathbf{M}_d(t) = \mathbf{f}\cdot \mathbf{M}_d(t)\ . \end{aligned} \qquad (1)$$

where the addition and multiplication operations are performed in $GF(2)$.[3] We have now expressed $f(\mathbf{K}_t)$as the product of $\mathbf{M}_d(t)$ with a row vector $\mathbf{f} = (0,0,0,1,0,0,0,0,0,1,0)$ that selects the values of the specific monomials required to evaluate the function f. *Note that the row vector* $\mathbf{f}$ *depends* ***only*** *on the function f(), and is independent of the LFSR feedback polynomial, the value of the initial state, and the index t.* ■

We know the mapping from one LFSR state to the next LFSR state. It is also possible to determine the mapping from one monomial state to the next monomial state.

Example 3. Consider a 4-bit LFSR as in Example 1 with monomial state

$$\mathbf{M}_d(t) = (1, k_0, k_1, k_2, k_3, k_0 k_1, k_0 k_2, k_0 k_3, k_1 k_2, k_1 k_3, k_2 k_3)^{\mathrm{T}}$$

If the LFSR has is of the form $s_{t+4} = s_{t+1} + s_t$, then the next state $\mathbf{K}_{t+1}$ has a corresponding next monomial state $\mathbf{M}_d(t+1)= (m'_0, m'_1, m'_2, m'_3, m'_4, m'_5, m'_6, m'_7, m'_8, m'_9, m'_{10})^{\mathrm{T}}$ which is related to the original monomial state as follows (only some relationships have been shown in order to save space):

$$\begin{aligned} m'_0 &= 1 & &= 1 & &= m_0 & &= (1,0,0,0,0,0,0,0,0,0,0)\cdot \mathbf{M}_d(t), \\ m'_1 &= k'_0 & &= k_1 & &= m_2 & &= (0,0,1,0,0,0,0,0,0,0,0)\cdot \mathbf{M}_d(t), \\ m'_4 &= k'_3 & &= k_0 + k_1 & &= m_1 + m_2 & &= (0,1,1,0,0,0,0,0,0,0,0)\cdot \mathbf{M}_d(t), \\ m'_{10} &= k'_2 k'_3 & &= k_3(k_0 + k_1) & &= m_7 + m_9 & &= (0,0,0,0,0,0,0,1,0,1,0)\cdot \mathbf{M}_d(t). \end{aligned}$$

Each component of the next monomial state is a linear function of the original monomial state. These linear functions for $m'_0, \ldots, m'_{D-1}$ can be combined into a matrix $\mathbf{R}_d$ (the "rewriting matrix") such that $\mathbf{M}_d(t+1) = \mathbf{R}_d \cdot \mathbf{M}_d(t)$. This matrix depends only of the LFSR and the degree d. ■

This example generalizes: for every LFSR and degree d there is a "*monomial state rewriting matrix*" $\mathbf{R}_d$ such that $\mathbf{M}_d(t+1) = \mathbf{R}_d \cdot \mathbf{M}_d(t)$**.** Moreover, for every t, the monomial state after t clocks of the LFSR can be expressed as a GF(2) matrix operation

$$\mathbf{M}_d(t) = \mathbf{R}_d^{\,t} \cdot \mathbf{M}_d, \qquad (2)$$

where $\mathbf{M}_d$ is the initial monomial state. Combining equations (1) and (2), we get another expression for $f(\mathbf{K}_t)$**:**

$$f(\mathbf{K}_t) = \mathbf{f}\cdot(\mathbf{R}_d^{\,t}\cdot \mathbf{M}_d) = (\mathbf{f}\cdot \mathbf{R}_d^{\,t})\cdot \mathbf{M}_d = \mathbf{f}_t \cdot \mathbf{M}_d\,, \qquad (3)$$

where: $\mathbf{f}_t$ =def $(\mathbf{f}\cdot \mathbf{R}_d^{\,t})$ depends solely on the function f , the monomial state update matrix $\mathbf{R}_d$ and the number of clocks t (all of which are known to the attacker); and $\mathbf{M}_d$ is the unknown initial state.

[3] All matrix multiplications in this paper are performed over GF(2), unless stated otherwise.

2.2 Algebraic Attacks

We always assume that the monomial state is unknown; it is the goal of algebraic attacks to determine the initial monomial state $\boldsymbol{M}_d$ (and thereby determine the initial LFSR state). The first step in an algebraic attack is to find a Boolean function h such that the equation

$$h(\boldsymbol{K}_t, z_t \ldots, z_{t+\theta}) = 0 \qquad (4)$$

is true for all clocks (or indices) t. The degree of h with respect to the bits of $\boldsymbol{K}_t$ we shall denote by d. Various methods have been proposed for finding such equations (see [AK03,CM03]). These equations typically have small values for θ. For simplicity we shall hereafter combine the keystream bit values $z_t \ldots, z_{t+\theta}$ into a *keystream vector* $\boldsymbol{z}_t$.

For the linearization approach, it is convenient to obtain an expression for $h(\boldsymbol{K}_t, \boldsymbol{z}_t)$ in terms of the keystream bits and bits of the initial monomial state $\boldsymbol{M}_d$. Based on equation (3) we can re-write

$$h(\boldsymbol{K}_t,\boldsymbol{z}_t) = \boldsymbol{h}(\boldsymbol{z}_t) \cdot (\boldsymbol{R}_d^t \cdot \boldsymbol{M}_d) = (\boldsymbol{h}(\boldsymbol{z}_t) \cdot \boldsymbol{R}_d^t) \cdot \boldsymbol{M}_d = \boldsymbol{h}_t(\boldsymbol{z}_t) \cdot \boldsymbol{M}_d$$

where the vector $\boldsymbol{h}_t(\boldsymbol{z}_t)$ =def $(\boldsymbol{h}(\boldsymbol{z}_t) \cdot \boldsymbol{R}_d^t)$ has components that depend on (a) the function h; (b) the monomial state rewriting matrix $\boldsymbol{R}_d$ associated with the monomials of degree of degree d or less; (c) the number of clocks t; and (d) the small keystream vector $\boldsymbol{z}_t$. Equation (4) is then transformed to:

$$\boldsymbol{h}_t(\boldsymbol{z}_t) \cdot \boldsymbol{M}_d = 0 \qquad (5)$$

The second step of an algebraic attack is to evaluate many vectors $\boldsymbol{h}_t(\boldsymbol{z}_t)$, $1 \le t \le D$ by substituting observed keystream vector $\boldsymbol{z}_t$. Each of the evaluated vectors $\boldsymbol{h}_t(\boldsymbol{z}_t)$ provides the attacker with a linear equation in the D unknown bits of the initial monomial state $\boldsymbol{M}_d$. Since there are D unknowns, around D linear equations will be required to obtain a solvable system. An initial choice of D equations may contain linearly dependent equations, so more than D equations may be required in order to get a completely solvable system. It is thought that not many more than D equations will be required in practice (see remark at the end of section 5.1 of [C03]), so we will assume D equations are sufficient.

The third step recovers $\boldsymbol{M}_d$ by solving the resulting system. The system can be solved by Gaussian elimination or more efficient methods [S69]. The complexity of solving such a system of equations is estimated to be $O(D^\omega)$ where ω (the Gaussian coefficient?) is estimated to be $\omega = 2.7$. In general, D will be about $C^n_d = O(n^d)$. This means the complexities of an algebraic attack are as follows:

- The complexity of finding the equation $h(\boldsymbol{K}_t,\boldsymbol{z}_t)$ depends on many factors and is beyond the scope of this paper.
- The amount of keystream required for the second step (the data complexity) is $O(D) = O(n^d)$.
- The complexity of the second step (substituting the keystream) is $O(D^2) = O(n^{2d})$, assuming that the functions $h(\boldsymbol{K}_t,\boldsymbol{z}_t)$ are relatively simple functions of the keystream; and
- the complexity of solving the system in the third step is $O(n^{2.7d})$.

Note 1. The complexity is exponential in the degree *d*. Hence, the low degree *d* is required for an efficient attack. Therefore, an attacker using an algebraic attack will always try to find a system of low degree equations. ■

2.3 Fast Algebraic Attacks

Courtois [C03] proposed "fast algebraic attacks", as a method for decreasing the degree of a given system of equations. For fast algebraic attacks, we presume that the function *H* can be written in the form

$$h(\boldsymbol{K}_t, z_t \ldots, z_{t+\theta}) \quad = \quad f(\boldsymbol{K}_t) + g(\boldsymbol{K}_b \boldsymbol{z}_t) \; = 0, \tag{6}$$

where *f* is of degree *d* in the bits of $\boldsymbol{K}_t$, and *g* is of degree $e < d$ in the bits of $\boldsymbol{K}_t$. Since the functions *f* and *g* are of two distinct degrees, it is simplest if we consider them as depending on distinct monomial states $\boldsymbol{M}_d$ and $\boldsymbol{M}_e$, with corresponding monomial state rewriting matrices $\boldsymbol{R}_d$ and $\boldsymbol{R}_e$. The number of monomials of degree *d* or less is $D = \sum_{i=0..d} C^n_i \sim C^n_d$, and the number of monomials of degree *d* or less is $E = \sum_{i=0..e} C^n_i \sim C^n_e$.

A fast algebraic attack obtains an advantage over the normal algebraic attacks by including an additional pre-computation step in which the attacker determines linear combinations of equation (6) that will cancel out the high-degree monomials of degree $\{e+1, e+2, \ldots, d\}$ that occur in $f(\boldsymbol{K}_t)$, but not in $g(\boldsymbol{K}_t \boldsymbol{z}_t)$.

First note that equation (3) allows $F(\boldsymbol{K}_t)$ and $G(\boldsymbol{K}_t \boldsymbol{z}_t)$ to be re-written as $f(\boldsymbol{K}_t) = \boldsymbol{f}_t \cdot \boldsymbol{M}_d$, and $g(\boldsymbol{K}_t \boldsymbol{z}_t) = \boldsymbol{g}_t(\boldsymbol{z}_t) \cdot \boldsymbol{M}_e$, where $\boldsymbol{f}_t = (\boldsymbol{f} \cdot \boldsymbol{R}_d^t)$ and $\boldsymbol{g}_t(\boldsymbol{z}_t) = (\boldsymbol{g}(\boldsymbol{z}_t) \cdot \boldsymbol{R}_e^t)$. Equation (6) is then transformed to:

$$\boldsymbol{f}_t \cdot \boldsymbol{M}_d + \boldsymbol{g}_t(\boldsymbol{z}_t) \cdot \boldsymbol{M}_e = 0. \tag{7}$$

In the fast algebraic attack pre-computation step, the attacker finds $(D+1)$ coefficients $b_0, \ldots, b_D \in \{0, 1\}$ such that

$$\sum_{i=0..D} b_i \boldsymbol{f}_{t+i} = 0, \; \forall \; t. \tag{8}$$

Equations (7) and (8) can be combined:

$$\sum_{i=0..D} b_i \, (\boldsymbol{f}_{t+i} \cdot \boldsymbol{M}_d + \boldsymbol{g}_{t+i}(\boldsymbol{z}_{t+i}) \cdot \boldsymbol{M}_e) = (\sum_{i=0..D} b_i \, \boldsymbol{g}_{t+i}(\boldsymbol{z}_{t+i})) \cdot \boldsymbol{M}_e$$

Thus, we obtain a linear expression in $\boldsymbol{M}_e$

$$\boldsymbol{g'}_t(\boldsymbol{z'}_t) \cdot \boldsymbol{M}_e \; = 0, \quad \text{where} \tag{9}$$

$$\boldsymbol{g'}_t(\boldsymbol{z'}_t) \qquad = \sum_{i=0..D} b_i \boldsymbol{g}_{t+i}(\boldsymbol{z}_{t+i}). \tag{10}$$

The second step of a fast algebraic attack is to evaluate many vectors $\boldsymbol{g'}_t(\boldsymbol{z'}_t)$, by substituting observed keystream vectors $\boldsymbol{z}_t$ in equation (10). Each of the evaluated vectors $\boldsymbol{g'}_t(\boldsymbol{z'}_t)$ provides the attacker with a linear equation in the *E* unknown bits of the initial monomial state $\boldsymbol{M}_e$. Equation (9) involves fewer unknowns than the initial equation (6) degree *d*, so fewer equations (9) are required in order to solve for the unknowns. *Reducing the number of unknowns and equations significantly improves the third step of the attack as solving the system of E equations (9) takes significantly less time than solving the system of D equations of (5).* Since there are only *E* unknowns in $\boldsymbol{M}_e$, the system can be solved with *E* equations. The complexity of the third step is now $O(E^\omega)$.

Courtois [C03] and Armknecht [A04] have proposed efficient methods for finding the coefficients of equation (8). The details are not relevant to this paper, but the complexities are provided in Table 2 for the purposes of comparison with the method proposed in Section 6 of this paper.

At this point, it is worth noting that evaluating $\boldsymbol{g'_t(z'_t)}$ (for each equation (9)) requires substituting the bits from the D keystream vectors $\{\boldsymbol{z}_{t+i}, 0 \le i \le D\}$. Obtaining E equations (9) can be achieved using the set of keystream vectors $\{\boldsymbol{z}_{t+i}, 0 \le i \le D, 1 \le t \le E\} = \{\boldsymbol{z}_{t+i}, 1 \le t \le D+E\}$.These keystream vectors can be obtained from the keystream bits z_{t+i}, $1 \le t \le D+E+\theta$; so the attack can be performed using as few as $(D+E+\theta) = O(D)$ keystream bits.

3 Substitution Complexity of Fast Algebraic attacks

Normal algebraic attacks and fast algebraic attacks differ in the complexity of substituting the keystream into the equations in step two. The function $h(\boldsymbol{z}_t)$ is a function of a small number of keystream bits z_{t+i}, $0 \le i \le \theta$; but the function $\boldsymbol{g'_t(z'_t)}$ is a function of a large number of keystream bits z_{t+i}, $0 \le t \le D+\theta$. This must be taken into account when determining the substitution complexity. The attacks in [A04,C03] underestimated the substitution complexity. The root cause appears to be a misunderstanding (discussed in the introduction).

The simple approach to substituting the keystream is to compute the values of $\boldsymbol{g_t(z_t)}$ and then substitute these values into the equations (10) individually.[4] Computing a single component of $(\boldsymbol{g'_t(z'_t)} = \sum_{i=0..D} b_i \boldsymbol{g_t(z_t)})$ for a single equation will require complexity $D/2$, since (on average) half of the coefficients b_i are expected to be zero. There are E terms, so the complexity of substituting the keystream into a single equation (10) is $E \times (D/2) = ED/2$. However, E equations are required in order to solve the system, and so the total cost of simple substitution will be $E \times (ED/2) = E^2D/2$. Table 3 lists the complexity of simple substitution for the fast algebraic attacks in the literature. *Note that the simple substitution significantly exceeds the complexity of solving the linear system of equations in these cases.*

Attack	n	d,e	D	E	*Run-time Complexity*			
					Claimed Subs $O(ED)$ WRONG	***Simple Subs*** $E^2D/2$	*Solving Linear System* E^ω	*Dominant term*
E0	$\mathbf{2^7}$	**4,3**	2^{23}	2^{18}	2^{41}	$\mathbf{2^{59}}$	2^{49}	$\mathbf{2^{59}}$
LILI-128	**89**	**4,2**	2^{21}	2^{12}	2^{33}	$\mathbf{2^{44}}$	2^{39}	$\mathbf{2^{44}}$
Toyocrypt	$\mathbf{2^7}$	**3,1**	2^{18}	2^7	2^{23}	$\mathbf{2^{31}}$	2^{20}	$\mathbf{2^{31}}$

Table 3. Comparing the complexity of simple substitution $O(E^2D)$ and the claimed complexity of substitution $O(ED)$.

[4] We ignore the cost of computing $\boldsymbol{g_t(z_t)}$ as this cost is independent of the cost of determining $\mathbf{g'_t(z'_t)}$ from the values of $\boldsymbol{g_t(z_t)}$.

4 The Discrete Fourier Transform

Suppose $f(t)$ is a Boolean function evaluated at P values $1 \le t \le P$. In our case, we are interested in the functions $f(\boldsymbol{K}_t)$ of the LFSR state $\boldsymbol{K}_t$. We want to look at a *discrete spectral analysis* of h.

Real Spectral Analysis: First, we'll consider a quick tangential topic. A common tool in analyzing a real-valued function $f(x)$ (such as a sound wave) evaluated on a domain $x \in [0,P]$ is to represents the function f as a sum of simple periodic functions (cosine and sine curves) where the function f is specified by the amplitudes of these periodic functions. That is, we represent f as a sum of cosine and sine curves

$$f(x) = A_0 + \sum_{\phi=1..\infty} A_\phi \cos((2\pi\phi/p) \cdot x) + \sum_{\phi=1..\infty} B_\phi \sin((2\pi\phi/p) \cdot x) .$$

with *amplitudes* A_ϕ and B_ϕ assigned for each *frequency* ϕ. The sequences $\{A_\phi\}$ and $\{B_\phi\}$ are the *Fourier series* for f, and evaluating the amplitudes is called a *spectral analysis*.

Discrete Spectral Analysis: To perform a spectral analysis of the discrete function $f(t)$ we use a similar idea to represent $f(t)$ using simple periodic functions defined as follows. Let λ be a value of order P in a field $\mathcal{F}$ that (a) contains GF(2) as a subfield and (b) has elements of order P. For example, we could use the complex field, then we would use $\lambda = e^{2\pi/P}$. Alternatively, we could use a Galois Field whose order is divisible by P and then choose a field element of order P as our value λ. The simple periodic functions that we use to represent $f(t)$ are the P functions $\Lambda_\phi(t) = \lambda^{\phi t}$, $0 \le \phi \le (P-1)$; these functions are analogous to the sine and cosine curves.

A discrete spectral analysis determines values in the chosen field $\mathcal{F}$ (e.g. complex field or Galois field) that correspond to "amplitudes" $F_\phi \in \mathcal{F}$. These amplitudes are to be assigned to the P periodic functions $\Lambda_\phi(t)$ so that

$$f(t) = \sum_{\phi=0..(P-1)} F_\phi \cdot \Lambda_\phi(t). \qquad (11)$$

In this way, $f(t)$ is represented as the sum of p periodic functions $\Lambda_\phi(t)$, $0 \le \phi \le (P-1)$. It is well known that the sequence $\{F_\phi\}$ can be computed directly from the sequence $\{f(t)\}$ as:

$$F_\phi = \sum_{t=1..P} f(t)/p \; \Lambda_\phi(-t). \qquad (12)$$

This calculation of $\{F_\phi\}$ from $\{f(t)\}$ as in (12) is called the *Discrete Fourier Transform* (DFT), while the calculation of $\{f(t)\}$ from $\{F_\phi\}$ is the Inverse DFT.

Properties: There are properties of the DFT that are relevant to this paper:

Linearity: If $c(t) = \alpha a(t) + \beta b(t)\ \forall t$, then $C_\phi = \alpha A_\phi + \beta B_\phi\ \forall \phi$.

The *convolution* of a and b of period p is another function c of period p with $c(t) = \sum_{i=1..P} a(t)b(i-t \pmod P),\ \forall t$. It is common to write $c = a * b$.

Convolution Property: If $c = a * b$ then $C_\phi = A_\phi B_\phi\ \forall \phi$.

Fast Fourier Transform: The most efficient method for computing the DFT, known as the Fast Fourier Transform (FFT) [CT65], requires a total of $P\log_2 P$ operations. The convolution takes P operations. There is also a Inverse FFT that takes the same amount of time to invert the DFT.

5 Applying the FFT to the Substitution Step

The inefficiency of the naïve substitution (in the second step) is indicated by two things:

- The equations (10) often re-use the same values of $\boldsymbol{g}_t(\boldsymbol{z}_t)$ when computing $\boldsymbol{g'}_t(\boldsymbol{z'}_t)$.
- The equations (10) all use the same linear combination;

This is the type of computation for which the FFT can offer significant advantages.

Suppose the attacker observes the sufficient amount of keystream, and evaluates the coefficients of each of the vectors $\boldsymbol{g}_t(\boldsymbol{z}_t)$ in equation (7) for $1 \le t \le D+E$. The attacker wants a fast way to determine the E vectors $\boldsymbol{g'}_t(\boldsymbol{z'}_t) = \sum_{i=0..D} b_i\, \boldsymbol{g}_{t+i}(\boldsymbol{z}_{t+i})$, $1 \le t \le E$, (see equation (10)) from the $(D+E)$ vectors $\boldsymbol{g}_t(\boldsymbol{z}_t)$.

Notation: We define a function sequence $\boldsymbol{g}(t) = \boldsymbol{g}_t(\boldsymbol{z}_t)$, $1 \le t \le D+E$, with $P = D+E$. We require a new discrete function β on the domain $1 \le t \le P$; defined with $\beta(t) = 0$, $1 \le t \le E-1$, and $\beta(t) = b_{p-t}$, $E \le t \le p$. The coefficients $b_0, b_1, \ldots, b_D$ appear in the reverse order at the end of the sequence $\{\beta(t)\}$, with the remainder of the sequence filled with zeroes. We chose this function because, on the domain $1 \le t \le E$, the convolution $(\boldsymbol{g}*\beta)$ has special properties. First,

$$\begin{aligned}(\boldsymbol{g}*\beta)(t) &= \sum_{i=1..p} \beta(i)\, \boldsymbol{g}(t-i \ (mod\ P)) \\ &= \sum_{i=1..E-1} \beta(i)\, \boldsymbol{g}(t-i \ (mod\ P)) + \sum_{i=E..p} \beta(i)\, \boldsymbol{g}(t-i \ (mod\ P)) \\ &= \sum_{i=E..p} b_{(p-i)}\, \boldsymbol{g}(t-i \ (mod\ P))\end{aligned}$$

Now substitute $j = (P-i)$, so the sum is performed over the domain $0 \le j \le D$. For these values of j, we notice that $t-i \equiv t-(P-j) \equiv t+j \pmod P$, and thus,

$$\begin{aligned}&\boldsymbol{g}(t-i \ (mod\ P)) = \boldsymbol{g}(t+j) = \boldsymbol{g}_{t+j}(\boldsymbol{z}_{t+j}). \\ \Rightarrow\quad &(\boldsymbol{g}(\boldsymbol{z})*\beta)(t) = \sum_{j=0..D} b_j\ \boldsymbol{g}_{t+j}(\boldsymbol{z}_{t+j}) = \boldsymbol{g'}_t(\boldsymbol{z'}_t)\end{aligned}$$

This is, the convolution contains the values of $\boldsymbol{g'}_t(\boldsymbol{z'}_t)$! Now, if we can find a fast way for computing the convolution $(\boldsymbol{g}*\beta)(t)$, we may have a faster method for computing $\boldsymbol{g'}_t(\boldsymbol{z'}_t)$ from the values of $\boldsymbol{g}_t(\boldsymbol{z}_t)$. This is where the Fast Fourier transform is useful.

For simplicity, denote the DFTs corresponding to $\boldsymbol{g}(t)$ and $\beta(t)$ by $\boldsymbol{G}_\phi$ and B_ϕ respectively. We are going to compute $\boldsymbol{g'}_t(\boldsymbol{z'}_t)$ by first computing $\boldsymbol{G}$ and B. We then exploit the convolution property which states that if we compute the values of $\boldsymbol{Q}_\phi = \boldsymbol{G}_\phi B_\phi$ and then apply the inverse DFT to $\boldsymbol{Q}$, then the resulting function $\boldsymbol{q}$ is the convolution of $\boldsymbol{g}(\boldsymbol{z})$ and β, that is $\boldsymbol{q}(t) = (\boldsymbol{g}*\beta)(t)$. The first E vectors $\boldsymbol{q}(t)$, $1 \le t \le E$, will correspond to the vectors $\boldsymbol{g'}_t(\boldsymbol{z'}_t)$; that is $\boldsymbol{g'}_t(\boldsymbol{z'}_t) = \boldsymbol{q}(t)$, $1 \le t \le E$.

This may seems like a strange way to compute $\boldsymbol{g'}_t(\boldsymbol{z'}_t)$, but it is very efficient when the FFT is applied:

- The DFT B of β can be pre-computed (since it is always the same): the FFT requires $(D+E)\log_2(D+E) \approx D \log_2 D$ complexity.
- The DFT $\boldsymbol{G}$ of $\boldsymbol{g}_t(\boldsymbol{z}_t)$ is computed by the FFT in time $ED \log_2 D$, since each of the E components must be computed individually.
- Now the DFTs $\boldsymbol{G}$ and B have been computed, $\boldsymbol{Q}$ can be computed from $\boldsymbol{G}$ and B by simply multiplying the corresponding coefficients together. This step requires time $(D+E)E$; it is smaller by a factor of about $\log_2 D$ than the FFT steps.

- Finally, $\boldsymbol{q}$ is obtained from $\boldsymbol{Q}$ by applying the Inverse FFT; this will take time $ED \log_2 D$.
- The appropriate vectors $\boldsymbol{g'_t(z'_t)}$, $1 \le t \le E$, are obtained directly from $\boldsymbol{q}$, $1 \le t \le E$.

The total complexity of computing the vectors $\boldsymbol{g'_t(z'_t)}$, $1 \le t \le E$, from the evaluations of the vectors $\boldsymbol{g_t(z_t)}$, $1 \le t \le (D+E)$ is:

$$ED \log_2 D + (D+E)E + ED \log_2 D \approx 2ED \log_2 D.$$

The improvement of this method over the naïve substitution is a factor of

$$E^2 D/2 \div (2ED \log_2 D)) = E / 4 \log_2 D$$

So the improvement depends on the size of E relative to $4\log_2 D$.

Table 3 shows the complexities of the FFT method for substitution for the current fast algebraic attacks in literature. In the case of $E0$, the improvement has been significant, and the substitution step no-longer dominates the run-time complexity. The improvement is less noticeable in the substitutions required for LILI-128 and Toyocrypt. The substitution step still comprises a significant portion of the complexity for these attacks.

Attack	D	E	Pre-Comp	Run-time Complexity					
				Data	Substitution			Solving System	Total
					Claimed ED	*Simple E^2D*	*FFT $2ED\log_2 D$*		
E0	2^{23}	2^{18}	2^{28}	2^{24}	2^{41}	2^{59}	$\mathbf{2^{47}}$	2^{49}	2^{49}
LILI-128	2^{21}	2^{12}	2^{26}	2^{60}	2^{33}	2^{44}	$\mathbf{2^{39}}$	2^{39}	$\mathbf{2^{40}}$
Toyocrypt	2^{18}	2^{7}	2^{23}	2^{18}	2^{23}	2^{31}	$\mathbf{2^{30}}$	2^{20}	$\mathbf{2^{30}}$

Table 3. Comparing the complexities $O(2ED\log_2 D)$ of substitution using the FFT method against the complexities of simple substitution $O(E^2D/2)$ and the claimed complexity of substitution $O(ED)$.

6 Improving the Pre-Computation Step

A Common Linear Combination: It is common knowledge that a square matrix satisfies its characteristic polynomial. That is, if $p(x) = \sum_{i=0..D} p_i \; x^i$ is the characteristic polynomial of $\boldsymbol{R}_d$, then

$$p(\boldsymbol{R}_d) = \sum_{i=0..D} p_i \, \boldsymbol{R}_d^{\,i} = \mathbf{0}, \qquad (13)$$

where $\mathbf{0}$ represents the all-zero matrix. Thus, if the coefficients $b_0,\ldots, b_D$ of (8) are assigned the values of the coefficients $p_0,\ldots, p_D$ of the characteristic polynomial of $\boldsymbol{R}_d$, then $(\sum_{i=0..D} b_i \, \boldsymbol{f}_{t+i}) = \sum_{i=0..D} (p_i \cdot \boldsymbol{f} \; \boldsymbol{R}_d^{\,t+i}) = \boldsymbol{f} \cdot \boldsymbol{R}_d^{\,t} \cdot (\sum_{i=0..D} p_i \, \boldsymbol{R}_d^{\,i}) = \boldsymbol{f} \cdot \boldsymbol{R}_d^{\,t} \cdot \mathbf{0} = 0$. So a logical choice for the coefficients $b_0,\ldots, b_D$ of (8) is the values the coefficients $p_0,\ldots, p_D$ of the characteristic polynomial of $\boldsymbol{R}_d$!

Note 2. The characteristic polynomial of $\boldsymbol{R}_d$ depends on the LFSR and the degree d, and is otherwise independent of the function f. This means that for all functions of degree d, the coefficients $p_0,\ldots, p_D$ (of the characteristic polynomial of $\boldsymbol{R}_d$) can be substituted into the equations (8) and (10)!

Direct Computation of the Linear Combination: Suppose an LFSR state $\boldsymbol{K}_t$ of length n is updated according to state update matrix L, and the characteristic polynomial of L is primitive.[5] The following theorem, while not explicitly stated by Key [K76], is a fairly obvious consequence of Key's ideas. We use the notation $w(\phi)$ to denote the Hamming weight of ϕ (that is, the number of 1's in the radix-2 representation of the integer ϕ).

Theorem 1. (*Largely due to Key* [K76]) A Boolean function $f(t)= f(\boldsymbol{K}_t)$ of degree d can be expressed as

$$f(t) = \sum_{\phi:\ w(\phi)\le d} F_\phi \cdot \Lambda_\phi(t) = \sum_{\phi:\ w(\phi)\le d} F_\phi \cdot (\lambda^\phi)^t \qquad (14)$$

where ϕ is an n-bit integer, and $\lambda \in GF(2^n)$ is a root of the characteristic polynomial of the LFSR state update matrix L. Moreover, the polynomial $p_d(x) = \prod_{\phi:\ w(\phi)\le d} (x - \lambda^\phi)$, is always a characteristic polynomial of $f(t)$. This polynomial is of degree $D=\sum_{i=0..d} C_n^{\ i}$. ■

In other words, the characteristic polynomial of $\boldsymbol{R}_d$ is $p_d(x) = \prod_{\phi:\ w(\phi)\le d} (x - \lambda^\phi)$. Note that there are D values ϕ with $w(\phi)\le d$, so the corresponding linear combination is the expected length for the sequence $\{f(t)\}$.

The authors are not aware of the best complexity of expanding $p_d(x)$. As a first upper bound, consider multiplying pairs of factors to obtain factors of degree 2, then multiply pairs of factors of degree 2 to get factors of degree 4, and so forth. The FFT can be applied to speed up the multiplication. At the j-th step, (starting at j=1) the factors are of degree $T=2^{j-1}$. To multiply two factors $a(x)= \sum_{i=0..T} a_i x^i$ and $b(x)= \sum_{i=0..T} b_i x^i$, first define two functions $\alpha(i)$ and $\beta(i)$ on the domain $[0,2T+1]$, with

$$\alpha(i) = a_i, \text{ and } \beta(i) = b_i, \qquad \text{for } 0\le i \le T+1,$$
$$\alpha(i) = 0, \text{ and } \beta(i) = 0, \qquad \text{for } T+2\le i \le 2T+1.$$

The convolution $\chi(i) = (\alpha * \beta)(i)$ contains the coefficients of the $c(x) = a(x) \times b(x)$. We know that the complexity of the convolution is $2 \cdot (2T) \cdot \log_2(2T)$. Since there will be $D/2T$ multiplications performed at each step (that is, for each j), the total complexity of this approach is:

$$\sum_{j=1..\log D} (D/2^i) \cdot 2 \cdot 2^j \cdot \log 2^j) = 2 \cdot \sum_{j=1..\log D} D \cdot j = 2D \cdot \sum_{j=1..\log D} j$$
$$= 2D \cdot (\log D \cdot (\log D - 1) / 2) \approx D (\log D)^2.$$

In Table 2, the complexity of this method is compared against the previous methods.

7 Conclusion

We have shown that some published "fast algebraic attacks" on stream ciphers underestimate the process complexity of one of the steps, and we provide correct complexity estimates for these cases. We then show an improved method, using Fast Fourier Transforms, for substituting keystream bits into the system of equations needing to be solved. We also made some observations about the linear combination used in the pre-computation step of the fast algebraic attack. In particular, we found the fastest

[5] This approach can be extended to cases where the characteristic polynomial is not primitive; for example, when the keystream is a function of more than one LFSR. See Key [K76] for more details.

known method for performing the pre-computation. The fast algebraic attack remains an extremely powerful technique for analyzing LFSR-based stream ciphers.

References

[A04] F. Armknecht: *Improving Fast Algebraic Attacks*, to be presented at FSE2004 Fast Software Encryption Workshop, Delhi, India, February 5-7, 2004.

[AK03] F. Armknecht and M. Krause: *Algebraic Attacks on Combiners with Memory*, proceedings of Crypto2003, Lecture Notes in Computer Science, vol. 2729, pp. 162-176, Springer 2003.

[B01] Bluetooth CIG, Specification of the Bluetooth system, Version 1.1, February 22, 2001. Available from www.bluetooth.com.

[CT65] Cooley, J. W. and Tukey, J. W., 1965, *An algorithm for the machine calculation of complex Fourier series*, Mathematics of Computation, **19**, *90*, pp. 297-301, 1990.

[C03] N. Courtois: *Fast Algebraic Attacks on Stream Ciphers with Linear Feedback*, Crypto2003, Lecture Notes in Computer Science, vol. 2729, pp. 177-194, Springer 2003.

[CM03] N. Courtois and W. Meier: *Algebraic Attacks on Stream Ciphers with Linear Feedback*, EUROCRYPT 2003, Warsaw, Poland, Lecture Notes in Computer Science, vol. 2656, pp. 345-359, Springer, 2003.

[K76] E. Key: *An Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators*, IEEE Transactions on Information Theory, vol. IT-22, No. 6, November 1976.

[M69] J. Massey: *Shift Register synthesis and BCH decoding*, IEE Transactions on Information Theory, IT-15 (1969), pp. 122-127.

[MI02] M. Mihaljević and H. Imai: *Cryptanalysis of Toyocrypt-HS1 stream cipher*, IEICE Transactions on Fundamentals, vol E85-A, pp. 66-73, January 2002. Available at www.csl.sony.co.jp/ATL/papers/IEICEjan02.pdf.

[R91] R. Rueppel: *Stream Ciphers;* Contemporary Cryptology: The Science of Information Integrity. G. Simmons ed., IEEE Press, New York, 1991.

[SDGM00] L. Simpson, E. Dawson, J. Golic and W. Millan: *LILI Keystream Generator;* Selected Areas in Cryptography SAC'2000, Lecture Notes in Computer Science, vol. 1807, Springer, pp. 392-407.

[S69] V. Strassen: *Gaussian Elimination is Not Optimal*; Numerische Mathematik, vol. 13, pp. 354-356, 1969.