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

Philip Hawkes[1] and Gregory G. Rose[1]

Qualcomm Australia, Level 3, 230 Victoria Rd, Gladesville, NSW 2111, Australia
{phawkes,ggr}@qualcomm.com

Abstract. Recently proposed algebraic attacks [2, 6] and fast algebraic attacks [1, 5] 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 [5] as a way of reducing run-time complexity by reducing the degree of the system of equations. Previous reports on fast algebraic attacks [1, 5] 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) [4] can be applied to decrease the complexity of the substitution step. Finally, it is shown 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. An explicit factorization of the corresponding characteristic polynomial 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) [11]. Such ciphers include *E0* [3], LILI-128 [12] and Toyocrypt(see [10]). 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 [5] and call it $\mathbf{K}$ 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

[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.

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 [2, 6] and fast algebraic attacks [1, 5]. Such attacks have provided the best analyses against some theoretical and deployed ciphers.

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 $\mathbf{K}$ and bits of the keystream $\mathbf{Z} = \{z_t\}$. Some methods [2, 6] 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 $\mathbf{K}$. The third step is to solve these algebraic equations to determine $\mathbf{K}$. This will be possible if the equations are of low degree in the bits of $\mathbf{K}$, 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 [5] 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 $\mathbf{K}$ and the keystream $\mathbf{Z}$ 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 $\mathbf{K}$. 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 [5] proposes using a method based on the Berlekamp-Massey algorithm [8] 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 [1] 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 [1, 5]) appear to have underestimated the complexity of substituting the keystream into the second system of equations.[2] The complexity was originally underestimated as only $O(DE)$ [5], 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 [1, 5]. However, simple substitution would require a complexity of $DE^2/2$ (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 $DE^2/2$, but in cases where E is large, the process complexity should still be considered $DE^2/2$ in the absence of specialized hardware. In many cases $DE^2/2$ actually exceeds the complexity of solving the system of equations, as shown in Table 1.[3] The second contribution of this paper is to show how the Fast Fourier Transform (FFT) [4] can be applied to decrease the complexity of the substitution step 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 Process Complexity
				Claimed (wrong)	*Simple*	*FFT*		
E0	2^{23}	2^{18}	2^{24}	2^{41}	2^{59}	2^{47}	2^{49}	2^{49}
LILI-128	2^{21}	2^{12}	2^{60}	2^{33}	2^{44}	2^{39}	2^{39}	2^{40}
Toyocrypt	2^{18}	2^{7}	2^{18}	2^{23}	2^{31}	2^{30}	2^{20}	2^{30}

Table 1. Comparison of substitution complexities for 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 make the observation that all functions of degree d satisfy a common function-independent linear combination of length $D = \sum_{i=0}^{d} \binom{n}{d}$ that is defined exclusively by the LFSR. Then we provide a direct method for computing this linear combination (based on the work of Key [7]). This method requires $c \cdot D(n(\log n)^2 + (\log_2 D)^3)$ operations for small constant c. This is a significant improvement on the complexities of previous methods.

This paper is organized as follows: Section 2 describes fast algebraic attacks. In Section 3 we discuss the complexity of substitution step for fast algebraic attacks. Section 4 reviews the Fast Fourier Transform and Section 5 describes how the FFT can speed up the substitution step. Section 6 contains some observations on the pre-computation step. Section 7 concludes the paper.

[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.

[3] The attack on LILI-128 requires only every $(2^{39} - 1)$-st bit from a keystream of length 2^{60}. The process complexity of selecting these bits is ignored in the literature, and could be an area for useful discussion.

Cipher	D	Courtois [5]: based on Berlekamp-Massey		Armknecht [1]	Direction Computation $c \cdot D(n(\log n)^2 + (\log_2 D)^3)$
		$C \cdot D(\log D)$	D^2	Parallel Method	
E0	2^{23}	$C \cdot 2^{28}$	2^{46}	2^{43}	2^{37}
LILI-128	2^{21}	$C \cdot 2^{26}$	2^{42}	-	2^{35}
Toyocrypt	2^{18}	$C \cdot 2^{23}$	2^{36}	-	2^{32}

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

2 Fast Algebraic Attacks

The length of the LFSR is n-bits; that is the internal state of the LFSR is $\mathbf{K}_t \in GF(2)^n$. A state $\mathbf{K}_{t+1}$ is derived from the previous state $\mathbf{K}_t$ by applying an (invertible) linear mapping $L : GF(2)^n \to GF(2)^n$, with $\mathbf{K}_{t+1} = L(\mathbf{K}_t)$. 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 $\mathbf{K}_t = L^t(\mathbf{K})$. 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 $\mathbf{K}_t$ and for a given degree d, we shall let $\mathbf{M}_d(t)$ (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

$D = \sum_{i=0}^{d} \binom{n}{i} \approx \binom{n}{d}$, so $\mathbf{M}_d(t)$ contains D components. The initial monomial state $\mathbf{M}_d$ corresponds to the initial state $\mathbf{K}$.

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

$$\begin{aligned}\mathbf{M}_d(t) &= (m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m10)^T \\ &= (1, k_0, k_1, k_2, k_3, k_0k_1, k_0k_2, k_0k_3, k_1k_2, k_1k_3, k_2k_3)^T,\end{aligned}$$

where "T" denotes the transpose of the matrix to make a column vector. For $\mathbf{K}_t = (0, 1, 1, 1)$, the values of the monomial components of $\mathbf{M}_d(t)$ are:

$$\begin{aligned}\mathbf{M}_d(t) &= (1, k_0 = 1, k_1 = 1, k_2 = 1, \ldots, k_1k_2 = 1, k_1k_3 = 0, k_2k_3 = 0)^T \\ &= (1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0)^T.\end{aligned}$$

The ordering of the monomial components is arbitrary; for consistency we will enumerate using lower subscripts first, as shown. □

Expressing Functions of the LFSR State. We can express any Boolean function of the LFSR state as a product of the matrix $\mathbf{M}_d(t)$ with a row vector.

Example 2. Consider a Boolean function of the state $f(\mathbf{K}_t) = k_2 + k_1k_3$ (using the LFSR state from Example 1). This function can be expressed:

$$\begin{aligned}f(\mathbf{K}_t) &= k_2 + k_1k_3 \\ &= 0 \times 1 \;+\; 0 \times k_0 \;+\; 0 \times k_1 \;+\; 1 \times k_2 \;+\; 0 \times k_3 \;+\; 0 \times k_0k_1 \\ &\quad + \; 0 \times k_0k_2 \;+\; 0 \times k_0k_3 \;+\; 0 \times k_1k_2 \;+\; 1 \times k_1k_3 \;+\; 0 \times k_2k_3 \\ &= (0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0) \cdot \mathbf{M}_d(t) = \mathbf{f} \cdot \mathbf{M}_d(t), \end{aligned} \tag{1}$$

where the addition and multiplication operations are performed in $GF(2)$. We have now expressed the Boolean function $f(\mathbf{K}_t)$ as the product of the matrix $\mathbf{M}_d(t)$ with a row vector

$$\mathbf{f} = (f_0, \ldots, f_{D-1}) = (0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0)$$

that selects the values of the specific monomials required to evaluate $f(\mathbf{K}_t)$. □

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.

The Monomial State Rewriting Matrix. The mapping from one LFSR state to the next LFSR state can be expressed as a matrix product $\mathbf{K}_{t+1} = L \cdot \mathbf{K}_t$. It is also possible to determine the mapping from one monomial state to the next monomial state as a matrix product $\mathbf{M}_d(t + 1) = \mathbf{R}_d \cdot \mathbf{M}_d(t)$.

Example 3. Consider a 4-bit LFSR as in Example 1 with monomial state $\mathbf{M}_d(t) = (1, k_0, k_1, k_2, k_3, k_0k_1, k_0k_2, k_0k_3, k_1k_2, k_1k_3, k_2k_3)^T$. 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 $\mathbf{M}_d(t + 1) = (m'_0, m'_1, m'_2, m'_3, m'_4, m'_5, m'_6, m'_7, m'_8, m'_9, m'_{10})^T$

which is related to the original monomial state as follows (only some relationships have been shown in order to save space):

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

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 $\mathbf{M}_d(t+1) = \mathbf{R}_d \cdot \mathbf{M}_d(t)$.

$$\mathbf{R}_d = \begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0 \\
0&0&1&0&0&0&0&0&0&0&0 \\
0&0&0&1&0&0&0&0&0&0&0 \\
0&0&0&0&1&0&0&0&0&0&0 \\
0&1&1&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&1&0&0 \\
0&0&0&0&0&0&0&0&0&1&0 \\
0&0&1&0&0&1&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&1 \\
0&0&0&0&0&0&1&0&1&0&0 \\
0&0&0&0&0&0&0&1&0&1&0
\end{bmatrix}$$

□

Notice that the matrix $\mathbf{R}_d$ depends only on 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

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

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

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

where the vector $\mathbf{f}(t) \stackrel{\text{def}}{=} \mathbf{f} \cdot \mathbf{R}_d^t$ 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). For example, the vectors $\mathbf{f}(t)$, $t \in \{0,1,2\}$ corresponding to the function f in Example 2 are:

$$\begin{array}{ll}
\mathbf{f}(0) = \mathbf{f} \cdot \mathbf{R}_d^0 = \mathbf{f} \cdot \mathbf{I} &= (0,0,0,1,0,0,0,0,0,1,0), \\
\mathbf{f}(1) = \mathbf{f} \cdot \mathbf{R}_d^1 = \mathbf{f}(0) \cdot \mathbf{R}_d &= (0,0,0,0,1,0,1,0,1,0,0), \\
\mathbf{f}(2) = \mathbf{f} \cdot \mathbf{R}_d^2 = \mathbf{f}(1) \cdot \mathbf{R}_d &= (0,1,1,0,0,0,0,0,0,1,1).
\end{array}$$

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 $\mathbf{M}_d$ (and thereby determine the initial LFSR state).

Step 1. The first step in an algebraic attack is to find a Boolean function h such that the equation

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

is true for all clocks (or indices) t. The degree of h with respect to the bits of $\mathbf{K}_t$ we shall denote by d. Various methods have been proposed for finding such equations (see [2, 6]). 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 $\mathbf{z}_t$.

For the linearization approach, it is convenient to obtain an expression for $h(\mathbf{K}_t, \mathbf{z}_t)$ in terms of keystream bits and bits of the initial monomial state $\mathbf{M}_d$.

1. Express $h(\mathbf{K}_t, \mathbf{z}_t)$ as the inner product of $\mathbf{M}_d(t)$ and a keystream-dependent vector $\mathbf{h}(\mathbf{z}_t)$:
$$h(\mathbf{K}_t, \mathbf{z}_t) = \mathbf{h}(\mathbf{z}_t) \cdot \mathbf{M}_d(t). \tag{5}$$
2. Now, Equation (2) can be substituted into Equation (5);
$$h(\mathbf{K}_t, \mathbf{z}_t) = \mathbf{h}(\mathbf{z}_t) \cdot (\mathbf{R}_d^t \cdot \mathbf{M}_d) = (\mathbf{h}(\mathbf{z}_t) \cdot \mathbf{R}_d^t) \cdot \mathbf{M}_d = \mathbf{h}(t) \cdot \mathbf{M}_d,$$
where $\mathbf{h}(t) \stackrel{\text{def}}{=} \mathbf{h}(\mathbf{z}_t) \cdot \mathbf{R}_d^t$.
3. Equation (4) is thereby transformed to the form:
$$\mathbf{h}(t) \cdot \mathbf{M}_d = 0. \tag{6}$$

The components of $\mathbf{h}(t) = (h_0(t), \ldots, h_{D-1}(t))$ depend on: (a) the function h; (b) the monomial state rewriting matrix $\mathbf{R}_d$ associated with the monomials of degree of degree d or less; (c) the number of clocks t; and (d) the small keystream vector $\mathbf{z}_t$. An attacker has access to all of this information, so the attacker is able to compute all of the components of $\mathbf{h}(t)$. This means that the only unknowns in Equation (6) are the components of the initial monomial state $\mathbf{M}_d$.

Step 2. The second step of an algebraic attack consists of substituting the observed keystream vector $\mathbf{z}_t$ into the components of the vectors $\mathbf{h}(\mathbf{z}_t)$ and then computing the vector $\mathbf{h}(t) = \mathbf{h}(\mathbf{z}_t) \cdot \mathbf{R}_d^t$. The vectors $\mathbf{h}(t)$ are evaluated for many indices t. Each of the evaluated vectors $\mathbf{h}(t)$ provides the attacker with a linear equation in the D unknown bits of the initial monomial state $\mathbf{M}_d$. 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 [5]), so we will assume D equations are sufficient.

Step 3. The third step recovers $\mathbf{M}_d$ by solving the resulting system of linear equations. The system can be solved by Gaussian elimination or more efficient methods [13]. The complexity of solving such a system of equations is estimated to be $O(D^\omega)$, where ω (known as the Gaussian coefficient) is estimated to be $\omega = 2.7$. In general, D will be about $\binom{n}{d} = O(n^d)$.

Complexities. The complexities of an algebraic attack are as follows:

- The complexity of finding the equation $h(\mathbf{K}_t, \mathbf{z}_t)$ 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 $D = O(n^d)$.
- The complexity of the second step (substituting the keystream into the equations) is $O(D^2) = O(n^{2d})$, assuming that the functions $h(\mathbf{K}_t, \mathbf{z}_t)$ are relatively simple functions of the keystream; and
- the complexity of solving the system in the third step is $O(n^{2.7}d)$.

Note 1. The complexity is exponential in the degree d. Hence, a 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 [5] 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

$$h(\mathbf{K}_t, z_t, \ldots, z_{t+\theta}) = u(\mathbf{K}_t) + v(\mathbf{K}_t, \mathbf{z}_t) \ = \ 0, \tag{7}$$

where u is of degree d in the bits of $\mathbf{K}_t$, v is of degree $e < d$ in the bits of $\mathbf{K}_t$ and *only* v *depends on the keystream.* Since the functions u and v are of two distinct degrees (in the bits of $\mathbf{K}_t$), it is simplest to consider them as depending on distinct monomial states $\mathbf{M}_d$ and $\mathbf{M}_e$, with corresponding monomial state rewriting matrices $\mathbf{R}_d$ and $\mathbf{R}_e$. There are $D = \sum_{i=0}^{d} \binom{n}{i} \approx \binom{n}{d}$ monomials of degree d or less, and $E = \sum_{i=0}^{e} \binom{n}{i} \approx \binom{n}{e}$ monomials of degree e or less .

A fast algebraic attack gains an advantage over the normal algebraic attacks by including an additional pre-computation step in which the attacker determines linear combinations of equation (7) that will cancel out the high-degree monomials of degree $e+1, e+2, \ldots, d$ that occur in $u(\mathbf{K}_t)$, but not in $v(\mathbf{K}_t, \mathbf{z}_t)$.

As in equation (3), $u(\mathbf{K}_t)$ and $v(\mathbf{K}_t, \mathbf{z}_t)$ are written as vector inner-products:

- $u(\mathbf{K}_t) = \mathbf{u} \cdot \mathbf{M}_d(t) = \mathbf{u}(t) \cdot \mathbf{M}_d$, where $\mathbf{u}(t) = (u \cdot \mathbf{R}_d^t)$ is a vector with D components (all of which are *independent* of the keystream); and
- $v(\mathbf{K}_t, \mathbf{z}_t) = \mathbf{v}(\mathbf{z}_t) \cdot \mathbf{M}_d(t) = \mathbf{v}(t) \cdot \mathbf{M}_e$, where $\mathbf{v}(t) = \mathbf{v}(\mathbf{z}_t) \cdot \mathbf{R}_e^t))$ is a vector with E components (some of which are *dependent* on the keystream).

Equation (7) is then transformed to:

$$\mathbf{u}(t) \cdot \mathbf{M}_d + \mathbf{v}(t) \cdot \mathbf{M}_e = 0. \tag{8}$$

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

$$\textstyle\sum_{i=0}^{D} b_i \cdot \mathbf{u}(t+i) = 0, \ \forall t. \tag{9}$$

Equations (8) and (9) can be combined:

$$\textstyle\sum_{i=0}^{D} b_i \cdot (\mathbf{u}(t+i) \cdot \mathbf{M}_d + \mathbf{v}(t+i) \cdot \mathbf{M}_e) = \left(\sum_{i=0}^{D} b_i \cdot \mathbf{v}(t+i)\right) \cdot \mathbf{M}_e \ .$$

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

$$\mathbf{v}'(t) \cdot \mathbf{M}_e = 0, \text{ where} \tag{10}$$

$$\mathbf{v}'(t) = \textstyle\sum_{i=0}^{D} b_i \cdot \mathbf{v}(t+i). \tag{11}$$

The second step of a fast algebraic attack is to evaluate many vectors $\mathbf{v}'(t+i)$, by substituting observed keystream vectors $\mathbf{z}_{t+i}$ into the vectors $\mathbf{v}(t+i)$ in equation (11). Each of the evaluated vectors $\mathbf{v}'(t)$ provides the attacker with a linear equation in the E unknown bits of the initial monomial state $\mathbf{M}_e$. Equation (10) involves fewer unknowns than the initial equation (7); this means that the fast algebraic attack requires fewer equations 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 (10) takes significantly less time than solving the system of D equations of (6).* The complexity of the third step is now $O(E^\omega)$.

Courtois [5] and Armknecht [1] have proposed efficient methods for finding the coefficients of equation (9). 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.

Data Complexity. Evaluating the vector $\mathbf{v}'(t)$ (for each equation (10)) requires substituting the bits from the D keystream vectors $\mathbf{z}_{t+i}$, $0 \le i \le D$. Obtaining E equations (10) can be achieved using the set of keystream vectors $\{\mathbf{z}_{t+i}, 0 \le i \le D, 1 \le t \le E\} = \{\mathbf{z}_t, 1 \le t \le D+E\}$. These keystream vectors can be obtained from the keystream bits $z_{t+i}, 1 \le t \le D+E+\theta$. Hence, 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 2. The vector $\mathbf{h}(t)$ is a function of a small number of keystream bits z_{t+i}, $0 \le i \le \theta$, but the vector $\mathbf{v}'(t)$ is a function of a large number of keystream bits z_{t+i}, $0 \le i \le D+\theta$.

As discussed in the introduction , a misunderstanding resulted the attacks $[1,5]$ failing to account for this difference.

The naïve approach to substituting the keystream is to compute the vectors $\mathbf{v}(t)$ first and then substitute these vectors into the equations (11) individually.[4] Computing a single component of the vector $\mathbf{v}'(t) = \sum_{i=0}^{D} b_i \mathbf{v}(t+i)$ for a single value of t will require complexity $D/2$, since (on average) half of the coefficients b_i are expected to be zero. There are E components in each vector $\mathbf{v}'(t)$, so the complexity of substituting the keystream to obtain a single vector $\mathbf{v}'(t)$ using equation (11) is $E \times (D/2) = ED/2$. That is, obtaining a single equation (11) has complexity is $ED/2$. Since E equations are required in order to solve the system, 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 simple substitution is significantly more complex than solving the linear system of equations in these cases.*

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

Table 3. Comparing the claimed complexities of substitution, the complexities of simple substitution and the complexity of solving the linear system.

4 The Discrete Fourier Transform

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

$$a(x) = A_0 + \textstyle\sum_{\phi=1}^{\infty} A_\phi \cdot \cos\left(\frac{2\pi\phi}{P} \cdot x\right) + \textstyle\sum_{\phi=1}^{\infty} A_\phi^* \cdot \sin\left(\frac{2\pi\phi}{P} \cdot x\right).$$

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

Discrete Spectral Analysis: Suppose $a(t)$ is a function defined at discrete values $t = \{1, 2, 3, \ldots, P\}$, and the values of $a(t)$ lie in a field $\mathcal{F}$. Such discrete

[4] We ignore the cost of computing $\mathbf{v}(t)$ as this cost is independent of the cost of determining $\mathbf{v}'(t)$ from the values of $\mathbf{v}(t)$.

functions are equivalent to sequences written $a = \{a(t)\}$. Discrete spectral analysis of a, like real spectral analysis, represents a using simple periodic sequences with period P. These periodic sequences are of the form $\Lambda_\phi = \{\Lambda_\phi(t) = \lambda^{\phi \cdot t}\}$, where $0 \leq \phi \leq (P-1)$, and λ is an element of multiplicative order P in some field $\mathcal{G}$; these functions are analogous to the sine and cosine curves.

In some cases, $\mathcal{F}$ has elements of multiplicative order P, and λ can be an element of $\mathcal{F}$; that is, $\mathcal{G} = \mathcal{F}$. In other cases, λ must be chosen in an a larger field $\mathcal{G}$ that is an extension field of $\mathcal{F}$. In either case, the field $\mathcal{G}$ is a vector space over $\mathcal{F}$; that is, elements of $\mathcal{G}$ are of the form $x = \sum_{i=1}^{p} x_i \nu_i$ for some basis $\{\nu_1, \ldots, \nu_p\}$, $p \geq 1$. Elements $x \in \mathcal{F}$ are mapped to elements $\overline{x} = x\overline{I} \in \mathcal{G}$ where $\overline{I}$ is the identity element of $\mathcal{G}$. Thus, the sequence a of elements of $\mathcal{F}$ is mapped to the sequence $\overline{a}$ with elements of $\mathcal{G}$. A discrete spectral analysis determines a sequence of P "amplitudes" $A = \{A_\phi \in \mathcal{G}, 0 \leq \phi \leq (P-1)\}$, such that the sequence $\overline{a}$, can be expressed as:

$$\overline{a}(t) = \sum_{\phi=0}^{P-1} A_\phi \cdot \Lambda_\phi(t). \tag{12}$$

In this way, each sequence value $\overline{a}(t)$ is represented as a linear combination of sum of P periodic sequences Λ_ϕ, $0 \leq \phi \leq (P-1)$. It is well known that the sequence of amplitudes A can be computed directly from the sequence $\overline{a}$ as:

$$A_\phi = \frac{1}{P} \cdot \sum_{t=1}^{P} \overline{a}(t) \cdot \Lambda_\phi(-t). \tag{13}$$

The calculation of A from $\overline{a}$ as in (13) is called the *Discrete Fourier Transform* (DFT), while the calculation of $\overline{a}$ from A is the Inverse DFT. The most efficient method for performing the DFT, known as the Fast Fourier Transform (FFT) [4], requires a total of $P \log_2 P$ operations in the field $\mathcal{G}$. There is also an Inverse FFT that uses the same amount of computation to invert the DFT.

Convolutions and the DFT. The convolution of two discrete sequences a and b of period P is another sequences y of period P with $y(t) = \sum_{i=1}^{P} a(t) \cdot b(i - t (\mathrm{mod}\ P))$, $\forall t$. These are sequences of elements from the field $\mathcal{F}$. It is common to write $y(t) = (a * b)(t)$. Computing the convolution according to first principles would take P^2 multiplication and addition operations in the field $\mathcal{F}$. However, the Convolution Property provides us with an alternative method.

Convolution Property $y(t) = (a * b)(t)$, $\forall t$ if and only if $Y_\phi = A_\phi \cdot B_\phi$, $\forall \phi$.

The convolution can be computed by applying the FFT to a and b to form A and B, forming $\{Y_\phi = A_\phi \cdot B_\phi\}$; and finally applying the inverse FFT to Y in order to form y. The total complexity is $3(P \log_2 P) + P = P(3 \log_2 P + 1)$ operations in the field $\mathcal{G}$. In the cases where $\mathcal{G} = \mathcal{F}$, the FFT method is faster by a factor of $P/(3 \log_2 P + 1)$. In other cases, computations in $\mathcal{G}$ cost more than computations in $\mathcal{F}$ and the advantage is less. This "trick" has been applied in many areas such as the fast multiplication of larger numbers and polynomials (the product of two polynomials is the convolution of the two corresponding sequences of coefficients). We shall use this trick in the next two sections.

5 Applying the FFT to the Substitution Step

The calculation in equation (11) is performed component-wise, so we will begin by focussing on the sequence of values for only one of the monomial components $v_\mu(t)$ of the vectors $\mathbf{v}(t) = (v_0(t), \ldots, v_{E-1}(t))$, $0 \leq \mu \leq (E-1)$, and the corresponding components $v'_\mu(t)$ of the vectors $\mathbf{v}'(t)$. Assume that the attacker has observed a sufficient amount of keystream, evaluated the values of $v_\mu(t)$ in equation (8) for $1 \leq t \leq (D+E)$, and determined the values $b_0, b_1, \ldots, b_D$, The attacker now needs fast way to determine the values $v'_\mu(t) = \sum_{i=0}^{D} b_i \cdot v_\mu(t+i)$, $1 \leq t \leq E$, (see equation (11)) from the values $v_\mu(t)$.

The inefficiency of using simple substitution is indicated by two things:

– Equations (11) often re-use the same values of $v_\mu(t)$ when computing $v'_\mu(t)$, $1 \leq t \leq E$.
– Equations (11) all use the same linear combination;

This problem appears similar to computing $(\beta * v_\mu)(t) = v'_\mu(t)$ for an appropriate sequence β. Indeed, if β is defined as $\beta(t) = 0$, $1 \leq t \leq P-D-1$, and $\beta(t) = b_{P-t}$, $P-D \leq t \leq P$, then $(\beta * v_\mu)(t) = v'_\mu(t)$ for $1 \leq t \leq P-D$, Thus, the FFT may be combined with the Convolution Property for computing $v'_\mu(t)$. The sequences v_μ and β are defined on the field $\mathcal{F} = GF(2)$, so $\mathcal{G}$ will be a field of the form $GF(2^p)$. We choose p to be the smallest value such that $2^p > (D+E)$, and define $P = 2^p - 1$. This choice seems best because it uses the smallest number of bits to represent elements of $\mathcal{G}$.

Basic DFT-based Substitution Algorithm

1. Map the sequences β and v_μ in $\mathcal{F}$ to sequences $\overline{\beta}$ and $\overline{v}_\mu$ in $\mathcal{G}$.
2. Apply the DFT to obtain the sequence of amplitudes B from $\overline{\beta}$.
3. Apply the DFT to obtain the sequence of amplitudes V from $\overline{v}_\mu$.
4. Compute $Q_\phi = B_\phi \cdot V_\phi$, $0 \leq \phi \leq (P-1)$.
5. Apply the inverse DFT to obtain $\overline{q}$ from Q. Note $\overline{q}(t) = (\overline{\beta} * \overline{v}_\mu)(t)$.
6. Extract $v'_\mu(t)$ from $\overline{q}(t)$, $1 \leq t \leq E$; $v'_\mu(t) = 1$ if $\overline{q}(t) = I$, else $v'_\mu(t) = 0$.

Complexity. This may seem like a strange way to compute $v'_\mu(t)$, but the algorithm is very efficient when the FFT is used to compute the DFTs:

– The values B_ϕ are computed via the FFT using $P(\log_2 P)$ field operations (operations in the field $\mathcal{G}$). For given values $b_0, b_1, \ldots, b_D$, the same sequence $\overline{\beta}$ is used for each monomial component and for each attacked keystream. The attacker should pre-compute and store B to save time.
– The values V_ϕ are computed via the FFT using $P(\log_2 P)$ field operations.
– The values $Q_\phi = V_\phi \cdot B_\phi$ are computing using $(D+E)$ field multiplications.
– The sequence $\overline{q}$ can be obtained from $\{Q_\phi\}$ by applying the standard Inverse FFT; this requires in time $P(\log_2 P)$ field operations.

The pre-computation of B requires $P(\log_2 P)$ field operations. The run-time total complexity for computing the value of $v'_\mu(t)$ from the values $v_\mu(t)$ is approximately $2P \log_2 P$ field operations. These field operations are more complex than $GF(2)$ (logical) operations. To a good approximation, each field operation is equal in complexity to $\log_2 P$ logical operations (much of this can be parallelized). Thus, for our calculations, the run-time complexity of the above algorithm is equivalent to around $2P(\log_2 P)^2$ logical operations.

Improvement 1. The above algorithm computes all $(D+E)$ values of $\overline{q}$, but only E of these values are ever used. An efficient alternative is to divide the linear combination into δ segments of length $D' = D/\delta$ and perform the FFTs on these segments using a smaller field $GF(2^{p'})$ where $2^{p'} - 1 = P' \geq E + D'$. If we define appropriate sequences $\beta[j]$, $1 \leq j \leq \delta$, then we may write:

$$v'_{\mu}(t) = \sum_{j=0}^{\delta-1} \sum_{i=0}^{D'} b_{D'j+i} \cdot v_{\mu}(t + D'j + i) = \sum_{j=0}^{\delta-1} (\beta[j] * v_{\mu}[j])(t).$$

with sub-sequences $\{v_{\mu}[j](t) = v_{\mu}(t + D'j)\}$. Now, $q(t) = \sum_{j=0}^{\delta-1} (\beta[j] * v_{\mu}[j])(t)$ if and only if $Q_{\phi} = \sum_{j=0}^{\delta-1} B[j]_{\phi} \cdot V[j]_{\phi}$. Thus, computing $q(t) = v'_{\mu}(t)$ requires: precomputing the FFTs of $\beta[j]$ computing the FFTS of $v_{\mu}[j]$; computing Q_{ϕ}; and applying the inverse FFT to Q to obtain q and thus $v'_{\mu}(t)$. The FFTs and Inverse FFT dominate the complexity, requiring $(\delta+1)P'(\log P')^2$ logical operations at run-time, where $P' \approx \frac{D}{\delta} + E$. The basic algorithm above uses $\delta = 1$. The optimal choice for δ (providing the lowest complexity) depends on D and E.

Improvement 2. The DFT-based substitution algorithm computes the values of $v'_{\mu}(t)$, $1 \leq t \leq E$, for only one component of the vectors $\mathbf{v}'(t)$, $1 \leq t \leq E$. There are a total of E monomial components $v'_{\mu}(t)$, $0 \leq \mu \leq E$ for each such vector; thus if each component is computed separately, then the total complexity of computing all components of every vector $\mathbf{v}'(t)$ would be approximately $E(\delta+1)P'(\log P')$ field operations, or $E(\delta+1)P'(logP')^2$ logical operations. Fortunately, p' monomial components can be packed into each computation. For a set $\{\mu_1, \ldots, \mu_p\}$ of monomial indices we define a sequence $\overline{v} \in \mathcal{G}$: $\overline{v}(t) = (v_{\mu'_p}(t), v_{\mu_{p'-1}}(t), \ldots, v_{\mu_1}(t))$. Then

$$\overline{v}'(t) = (\overline{\beta} * \overline{v})(t) = (v'_{\mu'_p}(t), v'_{\mu_{p'-1}}(t), \ldots, v'_{\mu_1}(t)),$$

provides p' values $v'_{\mu_i}(t)$ for the price of one, dividing the total complexity by p'.

Improved DFT-based Substitution Algorithm
Inputs: $v_{\mu_i}(t)$, $1 \leq t \leq (D+E)$, $1 \leq i \leq p'$; $B[j]$, $0 \leq j \leq \delta$.
Outputs: $v_{\mu_i}(t)$, $1 \leq t \leq E$, $1 \leq i \leq p'$.

1. Form $\overline{v}(t) = (v_{\mu'_p}(t), v_{\mu_{p'-1}}(t), \ldots, v_{\mu_1}(t))$, $1 \leq t \leq (D+E)$.
2. Form sub-sequences $\overline{v}[j] = \{\overline{v}(D'j+1), \ldots, \overline{v}(D'j+P')\}$, $0 \leq j \leq \delta$.
3. Apply DFT to obtain $V[j]$ from $\overline{v}[j]$, $0 \leq j \leq \delta$.
4. Compute $Q_{\phi} = \sum_{j=0}^{\delta-1} B[j]_{\phi} \cdot V[j]_{\phi}$, $0 \leq \phi \leq (P-1)$.
5. Apply inverse DFT to obtain $\overline{q}$ from Q .
6. Set $\overline{v'}(t) = \overline{q}(t)$, $1 \leq t \leq E$.
7. Output $\overline{v'}(t) = (v'_{\mu'_p}(t), v'_{\mu_{p-1}}(t), \ldots, v'_{\mu_1}(t))$, $1 \leq t \leq E$.

The run-time complexity, after applying these two improvements, is

$$(\delta+1)EP'(\log P')^2 \times \tfrac{E}{\log P'} = (\delta+1)EP'(\log P').$$

Table 4 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 in the substitution complexity is less noticeable required for LILI-128, and insignificant for Toyocrypt. The substitution step still comprises a significant portion of the complexity for these attacks.

In all cases, the first improvement did not affect the complexity significantly; the largest improvement was by a factor of 4. Interestingly, the optimal value of δ was $\delta \approx D/E$, for which $D' = E$ and $P' \approx 2E$. The corresponding complexity is around $D/E \cdot E \cdot 2E \cdot (\log_2 E + 1) \ \approx 2DE(\log_2 E)$

Cipher	D	E	Substitution			Solving System	Total
			Simple	FFT $(\delta+1)EP'(\log P')$			
			$E^2D/2$	Basic ($\delta = 1$)	Optimal Choice		
E0	2^{23}	2^{18}	$2^{59.2}$	2^{48}	$2^{46.8}$ ($\delta = 2^4$)	2^{49}	2^{49}
LILI-128	2^{21}	2^{12}	$2^{44.2}$	$2^{39.4}$	$2^{37.8}$ ($\delta = 2^8$)	2^{39}	2^{40}
Toyocrypt	2^{18}	2^7	$2^{31.4}$	$2^{31.2}$	$2^{29.2}$ ($\delta = 2^{10}$)	2^{20}	2^{29}

Table 4. Comparing the complexities of substitution using the FFT method against the complexities of simple substitution.

6 Improving the Pre-Computation Step

A square matrix satisfies its characteristic polynomial. That is, if $p^{(d)}(x) = \sum_{i=0}^{D} p_i x^i$ is the characteristic polynomial of $\mathbf{R}_d$, then

$$p^{(d)}(\mathbf{R}_d) = \sum_{i=0}^{D} p_i \cdot \mathbf{R}_d^i = \mathbf{0}, \tag{14}$$

where $\mathbf{0}$ represents the all-zero matrix. Suppose the coefficients $b_0, \ldots, b_D$ of Equation (9) are assigned the values of the coefficients $p_0, \ldots, p_D$ of the characteristic polynomial of $\mathbf{R}_d$. Then, for any function $u(\mathbf{K}_t)$ of degree d,

$$\sum_{i=0}^{D} b_i \cdot \mathbf{u}(t+i) = \sum_{i=0}^{D} p_i \cdot \left(\mathbf{u} \cdot \mathbf{R}_d^{t+i}\right) = \mathbf{u} \cdot \mathbf{R}_d^t \cdot \left(\sum_{i=0}^{D} p_i \cdot \mathbf{R}_d^i\right) = \mathbf{u} \cdot \mathbf{R}_d^t \cdot \mathbf{0} = 0.$$

The characteristic polynomial of $\mathbf{R}_d$ depends on the LFSR and the degree d, and is otherwise independent of the function $u(\mathbf{K}_t)$. Thus, the coefficients $p_0, \ldots, p_D$

(of the characteristic polynomial of $\mathbf{R}_d$) can be substituted for the coefficients $b_0, \ldots, b_D$ in equations (9) and (11) *for all functions* $u(\mathbf{K}_t)$ *of degree* d.

Most functions u of degree d have a minimal polynomial of degree $D = \sum_{i=0}^{d} \binom{n}{d}$ (see [9, Fact 6.55]); the minimal polynomial for these functions will be $p^{(d)}(x)$. However, there are functions the minimal polynomial is a smaller factor of $p^{(d)}(x)$. For example, in the attack on *E0* [1], $p^{(d)}(x)$ has length $D = 11,017,633$; while the minimal polynomial of the Boolean function used in the attack is of slightly smaller length $D' = 8,822,188$. Using $p^{(d)}(x)$ in the attack on *E0* (instead of using the minimal polynomial) would increase the complexity by a small amount. An advantage of the methods proposed in [1, 5] is that those method will find the minimal polynomial for a specified Boolean function, even if the minimal polynomial is smaller than $p^{(d)}(x)$.

It is not difficult to show that the polynomial $p^{(e)}(x)$ divides $p^{(d)}(x)$ This suggests that the linear combination $\sum_{i=0}^{D} p_i \cdot \mathbf{v}(t+i)$ may also be zero, thus resulting in a trivial equation $0 \cdot \mathbf{M}_e = 0$ that provides no information about the initial monomial state. The probability of this cancellation occurring is small; the vectors $\mathbf{v}(t+i) = \mathbf{v}(\mathbf{z}_{t+i}) \cdot \mathbf{R}_d^{t+i}$, in the sum $\sum_{i=0}^{D-E} p'_i \cdot \mathbf{v}(t+i) = \mathbf{v}'(t)$ depend on the keystream. Nonetheless, this suggests that a better approach would be to cancel only those components corresponding to monomials of degree greater than e using the polynomial

$$p^{(d/e)}(x) \stackrel{\text{def}}{=} p^{(d)}(x)/p^{(e)}(x) = \sum_{i=0}^{D-E} p'_i \cdot x^i.$$

The linear combination $\sum_{i=0}^{D-E} p' \cdot \mathbf{u}(t+i)$ cancels components corresponding to monomials of degree in the range $[e+1, d]$. but will not cancel components corresponding to monomials of degree e or less. Hence, $\mathbf{u}'(t) = \sum_{i=0}^{D-E} p' \cdot \mathbf{u}(t+i)$ can be considered as an E-dimensional vector $\mathbf{u}'(t) = \mathbf{u}' \cdot \mathbf{R}_e^t$ for some vector $\mathbf{u}'$. Equation 10 would then become

$$(\mathbf{u}'(t) + \mathbf{v}'(t)) \cdot \mathbf{R}_e = 0. \tag{15}$$

The probability that $\mathbf{u}'(t) = \mathbf{v}'(t)$ will depend on the probability that $\mathbf{u}' = \mathbf{v}(\mathbf{z})$ for a random $\mathbf{z}$. Unless $\mathbf{v}(\mathbf{z})$ is constant, this probability $\mathbf{u}' + \mathbf{v}(\mathbf{z}) = 0$ will be less than $1/2$. After substitution of many such vectors, the probability that $\mathbf{u}'(t) + \mathbf{v}'(t)$=0 will be very small and Equation (15) is highly unlikely to be trivial.

6.1 Direct Computation of the Linear Combination

Suppose an LFSR state $\mathbf{K}_t$ 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 [7], is a fairly obvious consequence of Key's ideas, so no proof is given. This result provides a direct method for computing $p^{(d)}(x)$.

[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 [7] for more details.

Theorem 1. *(Largely due to Key [7])* If $\gamma \in GF(2^n)$ is a root of the characteristic polynomial of the LFSR state update matrix, then the characteristic polynomial of $\mathbf{R}_d$ is $p^{(d)}(x) = \prod_{\psi : w(\psi) \le d} \left(x - \gamma^{\psi}\right)$, where $\psi \in GF(2^n)$ and $w(\psi)$ denotes the Hamming weight of ψ (that is, the number of 1's in the radix-2 representation of the integer ψ). $\square$

Factoring $p^{(d)}(x)$ into $GF(2)$ polynomials $y_\psi(x)$. Computing the entire product $\prod_{\psi : w(\psi) \le d} \left(x - \gamma^{\psi}\right)$ while in $GF(2^n)$ would be costly. Fortunately, the factors in $GF(2^n)$ can be easily grouped into $GF(2)$ polynomials of degree n or less. We define an equivalence relation "$\doteq$" where $\psi' \doteq \psi$ if and only if $\psi' \equiv 2^j \psi (\mathrm{mod}\ 2^n - 1)$, for some value j. Since the set $\{\psi' \doteq \psi\}$ is closed under multiplication by 2, the polynomial $y_\psi(x) = \prod_{\psi' \doteq \psi}(x - \gamma^{\psi'})$, has coefficients that are either 0 or the identity element I. That is, the product $y_\psi(x)$ can be represented as a $GF(2)$ polynomial. Thus, $p^{(d)}(x)$ can compute in two phases:

1. Compute the $GF(2)$ polynomials $y_\psi(x)$ for all ψ of weight d or less.
2. Multiply the $GF(2)$ polynomials $y_\psi(x)$ to form $p^{(d)}(x)$.

Computing $y_\psi(x)$. The FFT over $GF(2^n)$ may be used to compute the polynomials $y_\psi(x)$. First, apply the FFT to the sequences corresponding to $(x - \gamma^{\psi'})$ for $\psi' \doteq \psi$ to obtain sequences $\Gamma^{(\psi')}$, $\psi' \doteq \psi$. Second, form sequence Ω with $\Omega_\phi = \prod_{\psi' \doteq \psi} \Gamma_\phi^{(\psi')}$. Finally, apply the inverse FFT to Ω to obtain $y_\psi(x)$. The first step is the most costly; it requires $n(\log n)^2$ logical operations for each factor. There are D factors, so the total combined cost is $Dn(\log n)^2$ logical operations.

Multiplying $y_\psi(x)$ to form $p^{(d)}(x)$. The second phase has polynomials with coefficients in $GF(2)$ and uses FFT's in extension fields of $GF(2)$. Multiplying two $GF(2)$ polynomials in to get a product of degree less than $J = 2^j$ can be performed (via the FFT) using $J(1 + 3\log_2 J) = 2^j(1 + 3j)$ operations in the extension field $GF(2^j)$; this is equal to $2^j(1 + 3j)j$ logical operations.[6] Use the FFT to first multiply pairs of polynomials of degree n to get polynomials of degree $2n$. Then multiply pairs of polynomials of degree $2n$ to get polynomials of degree $4n$ and so forth until $p^{(d)}(x)$ is formed. The total complexity is

$$\sum_{j=\log_2 n}^{\log_2 D} \frac{D}{2^j} \cdot 2^j(1 + 3j)j = D \sum_{j=\log_2 n}^{\log_2 D} (3j^2 + j) \approx D(\log_2 D)^3.$$

The combined complexity for the two phases is $D[n(\log n)^2 + (\log_2 D)^3]$. 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

[6] The attacker can "pack" multiple $GF(2)$ polynomials into a single $GF(2^j)$ sequence and thereby compute the convolution of multiple pairs $GF(2)$ polynomials using the same amount of computation. This reduces complexity by a relatively small factor.

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

Acknowledgements. Acknowledgement is due to Nicolas Courtois for inspirational discussions and fine French cuisine during which the results in Section 6 came to light. The reviewers also provided many useful insights; in particular, the two improvements to the basic DFT-based substitution algorithm.

References

1. F. Armknecht: *Improving Fast Algebraic Attacks,* to be presented at FSE2004 Fast Software Encryption Workshop, Delhi, India, February 5-7, 2004.
2. 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.
3. Bluetooth CIG, *Specification of the Bluetooth system, Version 1.1, February 22, 2001.* Available from www.bluetooth.com.
4. J. W.Cooley and J. W. Tukey: *An algorithm for the machine calculation of complex Fourier series,* Mathematics of Computation, 19, 90, pp. 297-301, 1965.
5. N. Courtois: *Fast Algebraic Attacks on Stream Ciphers with Linear Feedback,* Crypto2003, Lecture Notes in Computer Science, vol. 2729, pp. 177-194, Springer 2003.
6. 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.
7. 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.
8. J. Massey: *Shift Register synthesis and BCH decoding,* IEEE Transactions on Information Theory, IT-15 (1969), pp. 122-127.
9. A. Menezes, P.van Oorschot and A. Vanstone *Handbook of Applied Cryptography,* CRC Press series on discrete mathematics and its applications, CRC Press LLC, 1997.
10. M. Mihaljevic 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.
11. R. Rueppel: *Stream Ciphers;* Contemporary Cryptology: The Science of Information Integrity. G. Simmons ed., IEEE Press, New York, 1991.
12. 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.
13. V. Strassen: *Gaussian Elimination is Not Optimal;* Numerische Mathematik, vol. 13, pp. 354-356, 1969.