Correlation Attacks on Stream Ciphers: Computing Low-Weight Parity Checks Based on Error-Correcting Codes

W T Penzhorn

Department of Electrical and Electronic Engineering
University of Pretoria, 0002 PRETORIA, South Africa
walter.penzhorn@ee.up.ac.za

Abstract. The fast correlation attack described by Meier and Staffelbach [6] on certain classes of stream ciphers, based on linear feedback shift registers, requires that the number of taps of the characteristic polynomial must be small – typically less than 10. The attack can be extended to characteristic polynomials with an arbitrary number of taps if it is possible to compute low-weight polynomial multiples of the feedback polynomial. In this paper we present an algorithm for the efficient computation of low-weight parity checks. The algorithm, based on the theory of cyclic block error-correcting codes, applies the ideas underlying majority-logic decoding of maximal-length codes. A statistical analysis shows that it is not realistic to consider weight–3 parity checks, and hence it is necessary to compute weight–4 parity checks. The proposed algorithm has a worst-case computational complexity of $O(2^{2k/3})$, which is essentially independent of the number of taps of the characteristic polynomial, and is suitable for linear feedback shift registers of approximately 100 bits.

1 Introduction

In secret-key cryptosystems, pseudonoise generators based on binary linear feedback shift-registers (LFSRs) are often used as running key generators. The required keystream $z = (z_j)$ is obtained by combining a fixed number of say, R, LFSRs by means of a combining function f, which is chosen to be non-linear in order to avoid cryptanalytic attacks using the Berlekamp-Massey algorithm [5]. For encryption, the plaintext sequence $m = (m_j)$ is added modulo 2 to the keystream sequence $z = (z_j)$ on a bit-by-bit basis to give the ciphertext sequence $c = (c_j)$.

The characteristic polynomial of each LFSR of length $k_i, i = 1, 2, \ldots, R$ is chosen to be *primitive*, and is assumed to be known to the analyst. Furthermore, it is assumed that the secret key of the cryptosystem specifies the initial states of each LFSR. The total number of keybits required to specify the initial states of the stream cipher generator is $\sum_{i=1}^{R} k_i$. In a brute force attack the $\prod_{i=1}^{R} 2^{k_i}$ possible states of the LFSRs will have to be examined, which is not feasible in practical systems. However, Siegenthaler [10] has shown that if there exists a measure of correlation between the keystream sequence and the outputs of the

LFSRs, it is possible to determine the initial state of each LFSR independently, thereby reducing the cryptanalytic attack to a divide-and-conquer attack, with approximate complexity $\sum_{i=1}^{R} 2^{k_i}$. Siegenthaler has successfully demonstrated the correlation attack for a number of combining functions proposed in the literature, viz. Brüer [1]Geffe [3]Pless [9]. Siegenthaler's attack amounts to an exhaustive search through the state space of each individual LFSR, and is feasible for values of k up to approximately 50.

Recently, it was shown by Meier and Staffelbach [6] that if the number of taps t of the characteristic polynomial is small it is possible to determine the initial LFSR states by means of an iterative algorithm, having complexity much less than an exhaustive search. The algorithm exploits the fact that the bits of a sequence generated by a LFSR satisfy a number of linear relationships, referred to as *parity check equations*, or simply *parity checks*.

The algorithm has asymptotic complexity $O(k)$, when the number of taps t is fixed. However, if t grows linearly with LFSR length k, the algorithm has complexity exponential in k. Consequently, the algorithm is only suitable for LFSRs with relatively few taps, i.e. parity checks having low weight $t < 10$. This is one of the most important factors which currently limits the effectiveness of the algorithm. Several researchers have proposed extensions of the original algorithm to overcome this limitation (see for example [2][7][11]).

In this paper we present a new method for the computation of low-weight parity checks based on the theory of error-correcting codes. The sequence generated by a LFSR is equivalent to the codeword of a maximal-length block code. By applying the ideas underlying majority-logic decoding of cyclic block codes we develop an efficient algorithm for the computation of low-weight parity-check equations, which can then be used in the Meier-Staffelbach algorithm. The novelty of this algorithm is that its computational complexity is essentially independent of the number of tap points of the characteristic polynomial of the LFSR which is being analysed.

The proposed algorithm can be used for the computation of weight-3 and weight-4 parity check equations. The proposed algorithm has an approximate computational complexity of $O(2^{2k/3})$, and judicious selection of parameters makes it feasible to compute low-weight parity checks for values of k up to approximately 100.

2 Review of the statistical model

Assume that a segment of T keystream digits is being observed by the cryptanalyst. From a practical viewpoint it is desirable that the value of T should be as small as possible; i.e. $T \ll N = 2^k - 1$. The LFSR sequence (a_j) is correlated with probability $q > 0.5$ to the keystream sequence, i.e.

$$P(z_j = a_j) = q > 0.5 \qquad j = 0, 1, 2, \ldots \tag{1}$$

We make the simplifying assumption that z_j depends only on the input a_j and the observation at time index j. The corruption of the LFSR sequence (a_j) due

to the other LFSRs in the stream cipher and the addition of the plaintext may be modelled by the addition of "error digits" e_j, which we assume to be i.i.d. variables, i.e. $z_j = a_j + e_j$, $j = 0, 1, 2, \ldots$. Hence

$$P(a_j = z_j) = P(e_j = 0) = q \qquad j = 0, 1, 2, \ldots \tag{2}$$

which is assumed to be the same for all indices j. The e_j may be expressed in terms of the *correlation coefficient* ξ,

$$P(e_j = 0) = q = \tfrac{1}{2}(1 + \xi) \tag{3}$$

The problem of the cryptanalyst is to restore the unknown LFSR sequence (a_j) from the observed keystream sequence (z_j). Meier and Staffelbach [6] first noted that this can be done efficiently by exploiting the linear relationships known to exist in a linear recurring sequence. The sequence (a_j) is produced by a LFSR with a primitive characteristic polynomial $p(x)$ of degree k having t non-zero terms,

$$p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_k x^k \tag{4}$$

with $c_0 = 1$ and $c_1, c_2, \ldots c_k \in \{0, 1\}$. The output sequence (a_j) is given by the linear recursion relation

$$a_j = c_1 a_{j-1} + c_2 a_{j-2} + \cdots + c_k a_{j-k} = \sum_{i=1}^{k} c_i a_{n-i} \qquad j = k, k+1, k+2, \ldots \tag{5}$$

The number of taps t of the LFSR is equal to the number of non-zero coefficients $\{c_1, c_2, \ldots, c_k\}$ of the characteristic polynomial $p(x)$. In the case of linear recurring sequences over $GF(2)$ the non-zero coefficients have values $c_i = 1$. Therefore, the linear relation described by (5) can be written as a parity check equation consisting of the $t + 1$ nonzero terms of the LFSR sequence a_j:

$$L = a_0 + a_1 + a_2 + \cdots + a_t = 0 \tag{6}$$

where the a_i denote those sequence digits multiplied by the nonzero coefficients c_i. Following [6], we can replace the bits of the LFSR sequence with the bits of the keystream sequence at the same index positions. Obviously, the parity check equation will not necessarily be satisfied.

As was noted in [6][2] the number of taps t of the characteristic polynomial, which determines the number of nonzero terms in a check equation L, strongly influence the computational complexity of the algorithm. It is easily shown that the probability of a parity check equation consisting of $t + 1$ non-zero terms is satisfied, is given by:

$$P(a_0 + a_1 + a_2 + \cdots + a_t = 0) = \tfrac{1}{2}(1 + \xi^{t+1}) \tag{7}$$

Therefore, in order to fully exploit the observed correlation between the LFSR sequence (a_j) and the keystream sequence (z_j), it is of crucial importance to

find low-weight parity checks, having a small number of non-zero terms. The purpose of this paper is to introduce a systematic procedure for obtaining low-weight parity checks.

3 Computation of low-weight checks based on error-correcting coding

We briefly recall some basic facts from the theory of algebraic error-correcting coding. For proofs and more details, see for example [4].

Definition 1. A linear, binary (n, k) block code over $GF(2)$ is a set of 2^k n-tuples with components taken from $GF(2)$. The 2^k n-tuples are commonly referred to as *codewords*, or *code vectors*. A linear, binary block code is a subspace of the vector space V_2^n of all n-tuples or binary vectors of length n.

A *generator matrix* G for an (n, k) linear block code $\mathcal{C}$ over $GF(2)$ is a $k \times n$ matrix whose rows are linearly independent and form the basis vectors for a linear subspace V_2^k of the n-dimensional vector space V_2^n. The 2^k codewords of $\mathcal{C}$ are all linear combinations over $GF(2)$ of the rows of G.

The *Hamming weight* $w(\boldsymbol{a})$ of a codeword $\boldsymbol{a}$ is equal to the number of nonzero digits in the codeword. The *Hamming distance* $d(\boldsymbol{a}, b)$ between two codewords $\boldsymbol{a}$ and b is equal to the number of coordinate positions in which they differ. For a linear code $\mathcal{C}$ the minimum distance d between any two distinct codewords $\boldsymbol{a}$ and b is given by

$$\begin{aligned} d &= \min\{d(\boldsymbol{a}, b) : \boldsymbol{a}, b \in \mathcal{C},\ \boldsymbol{a} \neq b\} \\ &= \min\{w(\boldsymbol{a})\} \end{aligned} \tag{8}$$

For any $k \times n$ generator matrix G with k linearly independent rows, there exists an $(n-k) \times n$ *parity-check matrix* H with $n-k$ linearly independent rows of length n. An n-tuple $\boldsymbol{a}$ is a codeword in the code generated by G if and only if $\boldsymbol{a} \cdot H = 0$. This implies that $G \cdot H = 0$.

If $\boldsymbol{a} = (a_0, a_1, \ldots, a_{n-1})$ and $b = (b_0, b_1, \ldots, b_{n-1})$ are binary vectors, their *scalar product* is given by:

$$\boldsymbol{a} \cdot b = a_0 b_0 + a_1 b_1 + \cdots + a_{n-1} b_{n-1} \tag{9}$$

If $\mathcal{C}$ is an (n, k) linear binary code, its *dual* or *orthogonal* code $\mathcal{C}^*$ is the set of vectors for which the scalar product is zero with respect to all the codewords in $\mathcal{C}$, i.e.

$$\mathcal{C}^* = \{\boldsymbol{a} \mid \boldsymbol{a} \cdot b = 0 \text{ for all } b \in \mathcal{C}\} \tag{10}$$

If $\mathcal{C}$ has generator matrix G and parity check matrix H, then $\mathcal{C}^*$ has generator matrix H, and parity check matrix G. Thus $\mathcal{C}^*$ is an $(n, n-k)$ code, and $\mathcal{C}^*$ is in the *orthogonal subspace* or *nullspace* of $\mathcal{C}$. Hence the dual code $\mathcal{C}^*$ consists of

2^{n-k} binary vectors taken from the vector space V_2^{n-k} which is the *null space* of V_2^k.

Each vector in V_2^n can be represented as a polynomial in x of degree less than or equal to $n-1$. The components of the vector form the coefficients of the polynomial, i.e. a codeword $\boldsymbol{a} = (a_0, a_1, \ldots, a_{n-1})$ corresponds to the polynomial $a(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$.

A (n, k) block code is called *cyclic* if, whenever $\boldsymbol{a} = (a_0, a_1, \ldots, a_{n-1})$ is in $\mathcal{C}$ then, then $\boldsymbol{a}' = (a_{n-1}, a_0, a_1, \ldots, a_{n-2})$ is also in $\mathcal{C}$. In polynomial representation a cyclic shift corresponds to $a_{n-1} + a_0 x + a_1 x^2 + \cdots + a_{n-2}x^{n-1}$ which is equivalent to $xa(x) \pmod{x^n + 1}$. Thus the codewords in $\mathcal{C}$ correspond to all the polynomials of degree less than n in the polynomial ring $\pmod{x^n + 1}$.

Every nontrivial cyclic (n, k) code $\mathcal{C}$ contains one codeword $g(x)$ of degree $n - k$, called the *generator polynomial*. The cyclic code $\mathcal{C}$ has the following generator matrix:

$$G = \begin{pmatrix} g(x) \\ xg(x) \\ \vdots \\ x^{k-1}g(x) \end{pmatrix} \tag{11}$$

All the other codewords in $\mathcal{C}$ are obtained by multiplying $g(x)$ by $q(x)$, where $q(x)$ is any polynomial of degree $k-1$ or less. For any codeword $a(x)$ in $\mathcal{C}$ holds:

$$a(x) = q(x)g(x) \equiv 0 \pmod{g(x)} \tag{12}$$

The generator polynomial $g(x)$ has the property that it divides $x^n + 1$, i.e. $g(x)h(x) = x^n + 1$. The polynomial $h(x)$ has degree k, and is commonly referred to as the *parity-check* polynomial. Each codeword $a(x)$ generated by $g(x)$ satisfies

$$a(x)h(x) \equiv 0 \pmod{x^n + 1}. \tag{13}$$

Let $\mathcal{C}$ be an (n, k) cyclic code with generator polynomial $g(x)$. Then its *dual code* $\mathcal{C}^*$ is also cyclic and is generated by the polynomial $g^*(x) = x^k h(x^{-1})$, where

$$h(x) = \frac{x^n + 1}{g(x)}. \tag{14}$$

4 Maximum-length codes

For our purposes it is important to note that a sequence of length $2^k - 1$ produced by a LFSR of length k, with characteristic polynomial $p(x)$, can be viewed as a codeword of the *maximum-length* block code. For any integer $k > 3$ there exists a nontrivial *maximum-length* block error-correcting code $\mathcal{C}$ with the following parameters:

- Block length: $n = 2^k - 1$

– Number of information digits: k

The maximum-length code $\mathcal{C}$ is a cyclic block code, described in terms of a primitive polynomial $p(x)$ of degree k. The generator polynomial for this code is given by $g(x) = x^n + 1/p^*(x)$, where $p^*(x)$ is the reciprocal of the parity polynomial $p(x)$, i.e. $p^*(x) = x^k p(x^{-1})$. Note that $p^*(x)$ is also a primitive polynomial of degree k.

The code $\mathcal{C}$ consists of the all-zero vector and $2^k - 1$ code vectors of weight 2^{k-1}. Due to its cyclic nature, the $2^k - 1$ non-zero code words of $\mathcal{C}$ are shifted versions of the generator polynomial $g(x)$.

From the viewpoint of stream cipher cryptanalysis it is important to note that the $2^k - 1$ nonzero codewords of $\mathcal{C}$ may be interpreted as the $2^k - 1$ output sequences which can be generated by a LFSR of length k. In this sense, the k-bit *information vector* which is encoded by $\mathcal{C}$ may be interpreted as the *secret key* which is to be used to initialise the states of the LFSR, and thereby to determine the output sequence which is generated by the LFSR.

The dual code of the maximum-length code $\mathcal{C}$ is a $(2^k - 1, 2^k - k - 1)$ cyclic block code $\mathcal{C}^*$, generated by the parity polynomial $p(x)$. It is easily shown that this dual code $\mathcal{C}^*$, which is the null space of the maximum-length code, is a *Hamming code*. It is a well-known fact that a Hamming code contains vectors of weight 3.

5 Orthogonal parity check sums for the maximum-length code

Our goal is to develop an efficient method for computing low-weight parity check equations which can be utilised in the Meier-Staffelbach algorithm. This method utilises the principles underlying majority-decoding of cyclic block codes.

Consider an (n, k) maximum-length code $\mathcal{C}$ with parity check matrix H. The row space of H is an $(n, n-k)$ Hamming, denoted by $\mathcal{C}^*$, which is the null space of $\mathcal{C}$. Note that for any code vector $\boldsymbol{a} \in \mathcal{C}$ and code vector $h \in \mathcal{C}^*$, the scalar product of $\boldsymbol{a}$ and h is zero, that is,

$$h \cdot \boldsymbol{a} = h_0 a_0 + h_1 a_1 + \cdots + h_{n-1} a_{n-1} = 0 \tag{15}$$

Equation (15) is called a *parity check* equation. Clearly, there are 2^{n-k} such parity check equations.

Now, suppose that a code vector $\boldsymbol{a} \in \mathcal{C}$ is transmitted over a binary symmetric channel. Due to channel noise, the transmitted vector is corrupted. Denote the additive binary noise vector as $e = (e_0, e_1, \ldots, e_{n-1})$ and let $z = (z_0, z_1, \ldots, z_{n-1})$ be the received vector. Then

$$z = \boldsymbol{a} + e. \tag{16}$$

For any vector $b \in \mathcal{C}^*$ we can form the following parity-check equation:

$$L = h \cdot z = h_0 z_0 + h_1 z_1 + \cdots + h_{n-1} z_{n-1} \tag{17}$$

If the received vector z is a code vector in $\mathcal{C}$, L must be zero; however, if z is not a valid code vector then L *may not* be zero. Combining (16) and (17) and using the fact that $h \cdot a = 0$, we obtain the following relationship between the check sum L and the error digits in e:

$$L = h \cdot z = h \cdot (a + e) = h \cdot e = h_0 e_0 + h_1 e_1 + \cdots h_{n-1} e_{n-1} \tag{18}$$

Definition 2. A set of parity check equations is said to be *orthogonal* on the i-th digit if x^i appears in each equation, but no x^j appears more than once in the set.

Now, consider the following set of distinct, orthogonal code polynomials:

$$\mathcal{S} = \{h(x) = x^i + x^j + x^{n-1} \mid 0 \leq i < j \leq n-1\} \tag{19}$$

where the $h_i(x)$ are taken from $\mathcal{C}^*$, the $(2^k - 1, 2^k - k - 1)$ Hamming code in the null space of $\mathcal{C}$. No two polynomials in $\mathcal{S}$ have any common terms except the term x^{n-1}. Otherwise, the sum of these two polynomials would be a code polynomial of only two terms in the Hamming code. This is impossible since the minimum weight of a Hamming code is 3. Therefore, the set $\mathcal{S}$ contains polynomials orthogonal on the highest-order digit position x^{n-1}. For each polynomial $h(x) \in \mathcal{S}$ holds:

$$h(x) = x^i + x^j + x^{n-1} \equiv 0 \bmod p(x) \tag{20}$$

To compute these $h(x)$, we start with a polynomial $x^{n-1} + x^j$ for $0 \leq j < n-1$, and then determine x^i such that $x^{n-1} + x^j + x^i$ is divisible by $p(x)$. This can be carried out as follows. Divide $x^{n-1} + x^j$ by $p(x)$ step by step, using long division until a single term x^i appears at the end of a certain step. Then $h(x) = x^{n-1} + x^j + x^i$ is a polynomial orthogonal on digit position x^{n-1}. Clearly, if we start with $x^{n-1} + x^i$, we would obtain the same polynomial $h(x)$. Thus, we can find $(n-1)/2 = 2^{k-1} - 1$ polynomials orthogonal on digit position x^{n-1}.

Example 1

Consider the $(15, 4)$ maximum-length block code $\mathcal{C}$, with $k = 4$ and block length $n = 2^4 - 1 = 15$, associated with the primitive polynomial $p(x) = 1 + x + x^4$. The parity polynomial for this code is given by $h(x) = p^*(x) = 1 + x^3 + x^4$, which is also primitive. The dual code $\mathcal{C}^*$ for this code is a $(15, 11)$ (cyclic) Hamming code, generated by the primitive polynomial $p(x) = 1 + x + x^4$. To obtain a parity check of weight 3 orthogonal on $x^{n-1} = x^{14}$, divide $x^{14} + x^{13}$ by $p(x) = 1 + x + x^4$ with long division:

$$
\begin{array}{r}
x^{10}+x^9+x^7+x^5+x^4+x^3+x^2 \\
x^4+x+1\,\overline{)\,x^{14}+x^{13}} \\
\underline{x^{14}\qquad\quad+x^{11}+x^{10}} \\
x^{13}+x^{11}+x^{10} \\
\underline{x^{13}\qquad\quad+x^{10}+x^9} \\
x^{11}\qquad\quad+x^9 \\
\underline{x^{11}\qquad\qquad\quad+x^8+x^7} \\
x^9+x^8+x^7 \\
\underline{x^9\qquad\qquad\quad+x^6+x^5} \\
x^8+x^7+x^6+x^5 \\
\underline{x^8\qquad\qquad\quad+x^5+x^4} \\
x^7+x^6\qquad\quad+x^4 \\
\underline{x^7\qquad\qquad\quad+x^4+x^3} \\
x^6\qquad\quad+x^3 \\
\underline{x^6\qquad\quad+x^3+x^2} \\
x^2\ (stop)
\end{array}
$$

As soon as the single term remainder x^2 appears, the division process is stopped. This gives the following relationship:

$$
\begin{aligned}
x^2+x^{13}+x^{14} &= (x^2+x^3+x^4+x^5+x^7+x^8+x^9+x^{10})(1+x+x^4) \\
&\equiv 0 \bmod (1+x+x^4)
\end{aligned} \tag{21}
$$

Therefore, $h_1(x) = x^2+x^{13}+x^{14}$ is a parity check polynomial orthogonal on x^{14}. To obtain another orthogonal polynomial, divide $x^{14}+x^{12}$ by $p(x)$. Then $h_2(x) = x^5+x^{12}+x^{14}$ is also a polynomial orthogonal on x^{14}. The remaining orthogonal polynomials can be found similarly. The $2^{4-1}-1 = 7$ polynomials are shown below:

$$
\begin{aligned}
h_1(x) &= x^2+x^{13}+x^{14} & h_5(x) &= x^6+x^8+x^{14} \\
h_2(x) &= x^5+x^{12}+x^{14} & h_6(x) &= x^1+x^7+x^{14} \\
h_3(x) &= x^{10}+x^{11}+x^{14} & h_7(x) &= 1+x^3+x^{14} \\
h_4(x) &= x^4+x^9+x^{14}
\end{aligned}
$$

6 Computing weight-3 polynomials

The computation of weight-3 parity checks orthogonal on x^{n-1}, by means of long division, can be readily applied to compute low-weight checks required for the Meier-Staffelbach algorithm. However, for the Meier-Staffelbach algorithm it is more convenient to have parity checks orthogonal on x^0 instead of x^{n-1}. By virtue of the fact that the parity equations $h_i(x)$ are codewords of a Hamming code, which is a *cyclic* code, every cyclic shift of a parity check equations produces

another valid parity check. For example, cyclically shifting $h_1(x)$, and reducing $\mathrm{mod}(x^{15}+1)$, gives:

$$\begin{aligned} h_1(x) &= x^2 + x^{13} + x^{14} \\ xh_1(x) &= x^3 + x^{14} + x^0 \\ x^2h_1(x) &= x^4 + x^0 + x^1 \end{aligned} \tag{22}$$

In a similar way, the set of parity check polynomials given in (5) which is orthogonal on x^{14}, may be shifted cyclically to obtain parity checks orthogonal on x^0. The shifts were done so as to obtain polynomials of lowest degree. The shifted, rearranged set is shown below:

$$\begin{aligned} &x^0 + x^1 + x^4 \\ &x^0 + x^2 + x^8 \\ &x^0 + x^3 + x^{14} \\ &x^0 + x^5 + x^{10} \\ &x^0 + x^6 + x^{13} \\ &x^0 + x^7 + x^9 \\ &x^0 + x^{11} + x^{12} \end{aligned} \tag{23}$$

In general, we are interested in obtaining parity checks orthogonal on *any* digit j of the LFSR sequence. The checks in (23) give rise to the following set of parity checks orthogonal on digit j:

$x^j + x^{j+1} + x^{j+4}$	$x^{j-1} + x^j + x^{j+3}$	$x^{j-4} + x^{j-3} + x^j$
$x^j + x^{j+2} + x^{j+8}$	$x^{j-2} + x^j + x^{j+6}$	$x^{j-8} + x^{j-6} + x^j$
$x^j + x^{j+3} + x^{j+14}$	$x^{j-3} + x^j + x^{j+11}$	$x^{j-14} + x^{j-11} + x^j$
$x^j + x^{j+1} + x^{j+5}$	$x^{j-5} + x^j + x^{j+5}$	$x^{j-10} + x^{j-5} + x^j$
$x^j + x^{j+6} + x^{j+13}$	$x^{j-6} + x^j + x^{j+7}$	$x^{j-13} + x^{j-7} + x^j$
$x^j + x^{j+7} + x^{j+9}$	$x^{j-7} + x^j + x^{j+2}$	$x^{j-9} + x^{j-2} + x^j$
$x^j + x^{j+11} + x^{j+12}$	$x^{j-11} + x^j + x^{j+1}$	$x^{j-12} + x^{j-1} + x^j$

Following Meier and Staffelbach [6], further parity check equations may be obtained by iterated squaring of the equations given in (23), for example:

$$\left[x^2h_1(x)\right]^2 = \left[1 + x + x^4\right]^2 = 1 + x^2 + x^8 \tag{24}$$

In this way a given weight-3 parity check equation can lead to several new parity checks, which are all orthogonal on x^0. The "toy example" we are considering here produces an unrealistically short sequence length of $2^4 - 1 = 15$. As a result, iterated squaring of the parity check equations in (23) does not lead to many useful equations, since the available sequence length is exceeded in most cases. However, in a real-life situation values of k in excess of say, 100, may be encountered. This give rise to a sequence length of $2^k - 1$, in which case iterated squaring will produce many useful equations. The division algorithm for computing weight-3 parity checks is briefly outlined below:

Division Algorithm

1. [Initialise] Choose a suitable value for the length of the subsequence $T < 2^k - 1$ available for analysis, and a value $\nu_{max} \leq T < 2^k - 1$ for the maximum degree of the weight-3 parity checks.
2. [Setup] Set $j = \nu_{max} - 1$.
3. [Division] Use long division to divide $x^{\nu_{max}} + x^j$ by $p(x)$ until a single-term remainder x^i is obtained. The required parity check is $x^i + x^j + x^{\nu_{max}}$.
4. [Exit] Set $j = j - 1$. If $j > 0$ return to Step 3. Else exit.

The division algorithm can be implemented very efficiently, since the required arithmetic operations essentially amount to shift and mod-2 addition operations. The algorithm may be further refined by observing that as the index j is stepped through all possible values, two sets of equivalent equations will be obtained, since $x^i + x^j + x^{\nu_{max}} = x^j + x^i + x^{\nu_{max}}$. Hence, the algorithm may be optimised by omitting the division operation for values of j equivalent to previously obtained values of i. Note that once a weight-3 parity check has been obtained, cyclic shifting produces a parity check orthogonal on any desired digit. Also, as illustrated above, iterated squaring will give other useful parity checks.

As seen in the preceding section, the total number of weight-3 parity checks available in a maximum-length (n, k) code is $2^{k-1} - 1$. The complexity of the long-division process to compute weight-3 checks from a sequence of length $N = 2^k - 1$ is roughly $O(2^{2k})$. However, in a practical situation the cryptanalyst will only have access to a subsequence of length $T \ll N$. In this case, the computational complexity of the division algorithm is approximately $O(T^2)$.

For practical application of the division algorithm to sequences generated by LFSRs of lengths of the order of 100, note that the length T of the subsequence may be chosen independently of the maximum degree ν_{max} of the parity checks in the set (23). The value of ν_{max} determines the number of division steps which the division algorithm will perform in search for a single-term remainder. On the other hand, the value of T essentially determines the number of parity checks of the form $x^{n-1} + x^j$ which will be processed by the division algorithm. The resulting computational complexity of the division algorithm is roughly $O(2^{T\nu_{max}})$. Provided that $\nu_{max} < T$, these two parameters can take on any suitable valued which will lead to practically feasible computation times. Hence it is of interest to obtain estimates for the values of ν_{max} and T which will produce weight-3 parity checks with high probability.

The long division algorithm determines a value r such that the following congruence is satisfied:

$$x^0 + x^i + x^r \equiv 0 \bmod p(x) \tag{25}$$

Observation of the behaviour of the algorithm for small values of 2^k indicates that the computed values r appear to be uniformly distributed across the interval $(0, N)$ where $N = 2^k - 1$ is the total length of the LFSR sequence.

Define a *hit* by the event that the computed value x^r falls within the subsequence available for cryptanalysis, i.e. $r \leq T$ and define a *miss* by the event

$r > T$. Assume that the number of hits in a subsequence follows a Poisson distribution with rate parameter $\lambda = T/N$.

Let $\nu_{\max}$ be the maximum degree of the weight-3 parity check polynomials. The probability that $r \leq \nu_{\max}$, under the negative exponential distribution assumption, is given by

$$P(r \leq \nu_{\max}) = 1 - e^{-T\nu_{\max}/N} \tag{26}$$

For $\nu_{\max} = O(2^{k/4})$ and a probability greater than $1 - e^{-1} \approx 63\%$ of finding a weight-3 polynomial, the length of the subsequence required by the cryptanalyst is $T = O(k^{3/4})$. A subsequence of this size would limit the technique to values of k less than approximately 60. Hence, we conclude that for large values of k the computation of weight-3 parity checks is not feasible. These observations were confirmed by computation of weight-3 parity checks for small values of k ($k \leq 16$).

However, this rather discouraging result should not prohibit us from applying the division algorithm to search for weight-3 parity-checks. Knowing that an exhaustive search has computational complexity $O(2^{T\nu_{max}})$, we may still select small values for these two parameters which result in feasible computation times. Obviously, the probability of finding weight-3 parity checks will be very low. However, once a weight-3 parity-check has been found, it is possible to derive several new parity checks by cyclic shifting and iterated squaring, as demonstrated in the previous section. Also, it should be kept in mind that the computational complexity of the division algorithm is essentially independent of the weight of the polynomial $p(x)$.

7 Computing weight-4 polynomials

The results of the previous section indicate that it is not computionally feasible to obtain a sufficient number of weight-3 parity checks with high probability. Hence we are forced to consider weight-4 polynomials. Fortunately, the division algorithm can be easily adapted to compute weight-4 parity-checks. The only difference is that a remainder consisting of *two* terms is required, instead of a single term remainder, as in the case of weight-3 parity checks. The advantage is that more than one weight-4 parity check will be found during the long division process. Hence, the division process has to be continued after the first equation is found. This is illustrated below.

Example 2

Consider again the $(15, 5)$ maximum-length code of the previous section, with parity polynomial $h(x) = p^*(x) = 1 + x^3 + x^4$. To obtain two parity check equations of weight 4 orthogonal on $x^{n-1} = x^{14}$, divide $x^{14} + x^{12}$ by $p(x) = 1 + x + x^4$ with long division:

$$
\begin{array}{r}
x^{10}+x^8+x^7+x^6+x^5+x \\
x^4+x+1\,\overline{)\,x^{14}+x^{12}} \\
\underline{x^{14}\qquad\quad+x^{11}+x^{10}} \\
x^{12}+x^{11}+x^{10} \\
\underline{x^{12}\qquad\qquad\quad+x^9+x^8} \\
x^{11}+x^{10}+x^9+x^8 \\
\underline{x^{11}\qquad\qquad\quad+x^8+x^7} \\
x^{10}+x^9\qquad+x^7 \\
\underline{x^{10}\qquad\qquad+x^7+x^6} \\
x^9\qquad\qquad+x^6 \\
\underline{x^9\qquad\qquad+x^6+x^5} \\
x^5 \\
\underline{x^5+x^2+x} \\
x^2+x\ (stop)
\end{array}
$$

In this way we obtain two weight-4 equations:

$$x^6+x^9+x^{12}+x^{14} \qquad\qquad x+x^2+x^{12}+x^{14}$$

These two equations are orthogonal on both x^{14} and x^{12}, which has some consequence for majority-logic decoding. However, in the case of the Meier-Staffelbach algorithm this should be of limited importance, since iterated squaring of these two equations will generate further equations, which can be selected so as to be orthogonal on x^{14}, viz. x^0.

Similar to the case of weight-3 polynomials, cyclic shifting produces parity checks of minimum degree which are orthogonal on x^0:

$$\begin{aligned} &x^0+x^3+x^6+x^8 \\ &x^0+x^2+x^4+x^5 \end{aligned} \tag{27}$$

Finally, the following set of parity checks orthogonal on an arbitrary digit x^j are obtained:

$$\begin{array}{ll}
x^j+x^{j+3}+x^{j+6}+x^{j+8} & x^j+x^{j+2}+x^{j+4}+x^{j+5} \\
x^{j-3}+x^j+x^{j+3}+x^{j+5} & x^{j-2}+x^j+x^{j+2}+x^{j+3} \\
x^{j-6}+x^{j-3}+x^j+x^{j+2} & x^{j-4}+x^{j-2}+x^j+x^{j+1} \\
x^{j-8}+x^{j-5}+x^{j-2}+x^j & x^{j-5}+x^{j-3}+x^{j-1}+x^j
\end{array}$$

Note that it is also possible to generate weight-4 parity checks by making use of available weight-3 parity checks, as follows. Consider two parity check equations orthogonal on x^0, with integers i, j and s, t such that $0 < i < j$ and $0 < s < t$:

$$\begin{aligned} &x^0+x^i+x^j \equiv 0 \pmod{p(x)} \\ &x^0+x^s+x^t \equiv 0 \pmod{p(x)} \end{aligned} \tag{28}$$

Assume that $j > s$. Then, multiplying (28) by $x^{j-s} = x^{\delta}$ and adding it to (28) a weight-4 equation is obtained:

$$x^0 + x^i + x^{j-s} + x^{t+j-s} = x^0 + x^i + x^{\delta} + x^{t+\delta} \tag{29}$$

Else, if $s > j$, let $\delta' = s - j$. Then

$$x^0 + x^t + x^{s-j} + x^{i+s-j} = x^0 + x^t + x^{\delta'} + x^{i+\delta'} \tag{30}$$

Thus, in either case, if $\max\{t + \delta,\ i + \delta'\} < T$, a useful weight-4 parity check orthogonal on x^0 has been found. We will make the assumption that if $|\delta| < T/2$, a usable relation results.

Assume that it is required to have m parity checks for the Meier-Staffelbach algorithm. Let r be the number of attempts to compute m parity checks. Under the assumption that the distance between the hits for weight-3 equations follows a negative exponential distribution, the probability p that two successive hits are within a distance $T/2$ as required for a weight-4 parity check, is $p = 1 - e^{-rT/2N} \approx rT/2N$ if $rT \ll N$. If r attempts are made, then the expected number of usable pairs are $pr = r^2T/2N \geq m$. A design with $N = O(N^{1/3})$ gives $r = O(N^{1/3})$. A given value of r determines, for a fixed characteristic polynomial, the length of the subsequence required for analysis.

Hence, if the cryptanalyst has access to a subsequence of ciphertext of length $N = 2^{k/3}$, with high probability a sufficient number of weight-4 parity checks will be found by means of the division algorithm. In this case the computational complexity of the division algorithm is approximately $O(2^{2k/3})$, and is essentially independent of the number of taps of the characteristic polynomial. Hence, this method is feasible for LFSRs lengths of approximately 100 bits.

8 Discussion

The computational complexity of *Algorithm B* suggested by Meier and Staffelbach [6] is largely due to their suggested use of "iterated squaring" for the computation of low-weight parity checks. In this paper we show how this limitation may be overcome by introducing a new algorithm for the computation of low-weight parity checks.

The algorithm is based on ideas for majority-logic decoding of block codes. A statistical analysis shows that if the cryptanalyst has access to a small subsequence of the total keystream sequence, the probability of obtaining a sufficient number of weight-3 parity checks is extremely low. Hence, one is compelled us to consider weight-4 parity checks. The proposed division algorithm is able to compute both weight–3 and weight–4 parity checks efficiently. If the cryptanalyst has access to a subsequence of ciphertext of length $2^{k/3}$, with high probability the division algorithm will yield a sufficient number of weight–4 parity checks. The worst-case computational complexity of the division algorithm for computing weight–4 parity checks is approximately $O(2^{2k/3})$, but may even be reduced

by judicious selection of various parameters. Thus, this method is feasible for LFSR lengths of approximately 100 bits.

An important attribute of the algorithm is that its computational complexity is essentially independent of the number of taps of the characteristic polynomial of the LFSR sequence. Thus, this method makes it possible to compute low-weight parity checks – and thus to apply the Meier-Staffelbach Algorithm B – to stream ciphers with arbitrary-weight characteristic polynomials. Hence, the security of a stream cipher system cannot be guaranteed by the use of characteristic polynomials with a large number of taps. Consequently, if a stream cipher comprises LFSRs of small length ($k < 100$, say), utmost care should be exercised by the designer to ensure that the correlation between ciphertext and LFSR output is reduced to a minimum.

Acknowledgement

We would like to thank Professor J.L. Massey of the ETH in Zürich, Switzerland, for suggesting the use of majority-logic decoding techniques to compute low-weight parity checks.

References

1. J. O. Brüer, "On nonlinear combinations of linear shift register sequences", in *Proc. IEEE ISIT*, les Arcs, France, June 21-25 1982.
2. C. Chepyzhov and B. Smeets, "On a fast correlation attack on stream ciphers", in *Advances in Cryptology – EUROCRYPT '91.* 1991, ypp. 176–185, Springer-Verlag.
3. P. R. Geffe, "How to protect data with ciphers that are really hard to break", *Electronics*, pp. 99–101, January 1973.
4. S. Lin and D. Costello, **Error control coding: Fundamentals and applications**, Prentice-Hall, 1983.
5. J. L. Massey, "Shift–register synthesis and BCH decoding", *IEEE Trans. Information Theory*, vol. IT-15, pp. 122–127, 1969.
6. W. Meier and O. Staffelbach, "Fast correlation attacks on certain stream ciphers", *Journal of Cryptology*, vol. 1, no. 3, pp. 159–176, 1989.
7. M. J. Mihaljevic and J. Golic, "A comparison of cryptanalytic principles based on iterative error-correction", in *Advances in Cryptology – EUROCRYPT '91.* 1991, pp. 527–531, Springer-Verlag.
8. K. Nishimura and M. Sibuya, "Probability to meet in the middle", *J. of Cryptology*, vol. 2, no. 1, pp. 13–22, 1990.
9. V. S. Pless, "Encryption schemes for computer confidentiality", *IEEE Trans. Computers*, vol. C-26, pp. 1133–1136, November 1977.
10. T. Siegenthaler, "Decrypting a class of stream ciphers using ciphertext only", *IEEE Trans. Computers*, vol. C-34, pp. 81–85, 1985.
11. K. Zeng and M. Huang, "On the linear syndrome method in cryptanalysis", in *Advances in Cryptology – CRYPTO '88.* 1990, pp. 469–478, Springer-Verlag.