How to determine linear complexity and k-error linear complexity in some classes of linear recurring sequences

Wilfried Meidl

Sabancı University, MDBF, Orhanlı, Tuzla, 34956 İstanbul, Turkey
email: wmeidl@sabanciuniv.edu; Tel: +90 216 4839583; Fax: +90 216 4839550

Abstract

Several fast algorithms for the determination of the linear complexity of d-periodic sequences over a finite field $\mathbb{F}_q$, i.e. sequences with characteristic polynomial $f(x) = x^d - 1$, have been proposed in the literature. In this contribution fast algorithms for determining the linear complexity of binary sequences with characteristic polynomial $f(x) = (x-1)^d$ for an arbitrary positive integer d, and $f(x) = (x^2+x+1)^{2^v}$ are presented. The result is then utilized to establish a fast algorithm for determining the k-error linear complexity of binary sequences with characteristic polynomial $(x^2+x+1)^{2^v}$.

Keywords Linear complexity, k-error linear complexity, algorithm, linear recurring sequences, stream cipher

AMS Classifications 94A55, 94A60, 11B50

1 Introduction

A sequence $S = s_0, s_1, s_2, \ldots$ with terms in a finite field $\mathbb{F}_q$ with q elements is called a *linear recurring sequence* over $\mathbb{F}_q$ with *characteristic polynomial*

$$f(x) = \sum_{i=0}^{d} c_i x^i \in \mathbb{F}_q[x]$$

of degree d, if

$$\sum_{i=0}^{d} c_i s_{n+i} = 0 \quad \text{for } n = 0, 1, \ldots.$$

Without loss of generality we can always assume that $f(x)$ is monic, i.e. $c_d = 1$. If additionally $c_0 \neq 0$ then the recurring sequence is purely periodic. For a comprehensive disquisition of linear recurring sequences we refer to [11, Chapter 8].

Let S be a linear recurring sequence over $\mathbb{F}_q$, then its *minimal polynomial* is defined to be the (uniquely determined) monic polynomial $m(x) \in \mathbb{F}_q[x]$ of smallest degree such that $m(x)$ is a characteristic polynomial for S. The degree of $m(x)$ is called the *linear complexity* $L(S)$ of the sequence S. In engineering terms, $L(S)$ is the length of the shortest linear feedback shift register that can generate S, with the convention that $L(S) = 0$ if S is the zero sequence. Linear feedback shift registers are frequently used as building blocks of keystream generators for stream ciphers. As a consequence, insight in the nature of linear complexity is valuable in the analysis of the security of stream ciphers.

A linear recurring sequence S with characteristic polynomial $f(x)$ of degree d is completely determined by f and the first d terms of S. Consequently we can identify S with the d-tuple $(s_0, s_1, \ldots, s_{d-1})$. Clearly this first d terms also fix the minimal polynomial of the sequence S with characteristic polynomial $f(x)$ of degree d, and thus its linear complexity.
We will follow the notation in [8] and denote the set of all linear recurring sequences with characteristic polynomial $f(x)$ by $\mathcal{M}(f)$.

Extensive research has been carried out on linear recurring sequences with characteristic polynomial of the form $f(x) = x^d - 1$ (see [4, 7, 14, 15, 16]). In this case the first d terms exactly correspond with a period of the sequence. For several types of integers d and finite fields $\mathbb{F}_q$ fast algorithms that determine the linear complexity of a sequence $S \in \mathcal{M}(x^d - 1)$ over $\mathbb{F}_q$ have been proposed in the literature ([1, 2, 9, 13, 20, 21]). In this contribution we initiate the construction of algorithms for the calculation of the linear complexity in the more general viewpoint of sequences in $\mathcal{M}(f)$ for arbitrary monic polynomials $f(x)$. In Section 2 we present some necessary background and state a simple generalization of the Games-Chan algorithm [9] for binary sequences in $\mathcal{M}((x-1)^d)$ for an arbitrary positive integer d. In Section 3 we establish an algorithm which determines the linear complexity of binary sequences in $\mathcal{M}((x^2+x+1)^{2^v})$, $v \geq 1$.

A related complexity measure is the *k-error linear complexity*. The k-error linear complexity of a d-periodic sequence S over $\mathbb{F}_q$ (i.e. a linear recurring sequence with characteristic polynomial $f(x) = x^d - 1 \in \mathbb{F}_q[x]$) is defined to be the smallest value to which the linear complexity of S can be reduced by performing k or fewer term changes in the first period (i.e. among the first d terms) of S and identical changes in every period thereafter (see [19], and [6] for the closely related and even earlier defined *sphere complexity* and *weight complexity* of S).

In Section 4 we adapt the concept of the k-error linear complexity to sequences in $\mathcal{M}(f)$ and present an algorithm to determine the k-error linear complexity (spectrum) of binary sequences in $\mathcal{M}((x^2+x+1)^{2^v})$, $v \geq 1$.

Algorithms for determining linear complexity and k-error linear complexity are useful in the analysis of the security of stream ciphers. However it has to be pointed out that our algorithms, just as the Games-Chan algorithm and the algorithms in [20, 21], require the knowledge of d terms of the sequence $S \in \mathcal{M}(f)$ if $\deg(f) = d$, which limits the applicability of the algorithms in classical cryptanalysis scenario where an attacker wants to recover the whole sequence with the knowledge of a limited amount of consecutive terms of the sequence. (For a short discussion on this disadvantage we can refer to the introductions of [10, 18]).

2 Preliminaries

Given a sequence $S = s_0, s_1, s_2, \ldots$ over $\mathbb{F}_q$ we can associate it with its *generating function* $S(x) = s_0 + s_1 x + s_2 x^2 + \cdots = \sum_{i=0}^{\infty} s_i x^i$. If S is a linear recurring sequence with characteristic polynomial $f(x)$ of degree d, then there exists a unique polynomial $g(x)$ of degree at most $d-1$ such that

$$S(x) = \frac{g(x)}{f^*(x)}, \tag{1}$$

where $f^*(x)$ denotes the reciprocal of the characteristic polynomial $f(x)$, i.e. $f^*(x) = x^d f(1/x)$. Moreover equation (1) describes a one to one correspondence between the set of polynomials of degree less than d and the set $\mathcal{M}(f)$ (cf. [11, Section 8], [17]). The polynomial $g(x)$ corresponding to a sequence $S \in \mathcal{M}(f)$ can obviously be calculated from the first d terms of S with equation (1). We remark that $g(x) = s_0 + s_1 x + \cdots + s_{d-1}x^{d-1}$ if $f(x) = x^d - 1$, i.e. $\mathcal{M}(f)$ is the set of all d-periodic sequences over $\mathbb{F}_q$.

Let $S(x) = g(x)/f^*(x)$ be the generating function of a sequence $S \in \mathcal{M}(f)$, then

$$S(x) = \frac{g(x)}{f^*(x)} = \frac{g(x)/\gcd(g(x), f^*(x))}{f^*(x)/\gcd(g(x), f^*(x))} := \frac{h(x)}{m^*(x)},$$

where $m^*(x)$ is the reciprocal of the minimal polynomial $m(x)$ of the sequence S. By well known properties of reciprocals, $m(x)$ is a divisor of $f(x)$. The linear complexity of S is consequently given by (cf. [3, Lemma 8.2.1], [8])

$$L(S) = \deg(f) - \deg(\gcd(g(x), f^*(x))). \tag{2}$$

For further information on linear recurring sequences we again refer to [11].

In one of the earliest contributions on determining the linear complexity of sequences with characteristic polynomial of the form $x^d - 1$ Games and Chan [9] presented an algorithm which efficiently determines the linear complexity of a binary sequence S with characteristic polynomial $f(x) = x^{2^v} - 1 = (x-1)^{2^v}$, i.e S is 2^v-periodic. The algorithm is based on the observation that the linear complexity $L(S)$ of a binary sequence $S = (s_0, s_1, \ldots, s_{2^v-1})$ of $\mathcal{M}(x^{2^v} - 1)$ equals $2^{v-1} + L(S_1)$, where S_1 is the sequence in $\mathcal{M}(x^{2^{v-1}} - 1)$ given by $S_1 = (s_0 + s_{2^{v-1}}, s_1 + s_{2^{v-1}+1}, \ldots, s_{2^{v-1}-1} + s_{2^v-1})$, unless $S \in \mathcal{M}(x^{2^{v-1}} - 1)$. In the second case S is of course treated as binary 2^{v-1}-periodic sequence, i.e. as binary sequence with characteristic polynomial $x^{2^{v-1}} - 1$. Applying this observation recursively one obtains the Games-Chan algorithm [9].

With some simple arguments one can generalize the Games-Chan algorithm for binary sequences in $\mathcal{M}(x^{2^v} - 1)$ to an algorithm for binary sequences in $\mathcal{M}((x-1)^d)$ for arbitrary integers d. Suppose that $2^v < d < 2^{v+1}$, then we certainly could use the first d bits of a sequence $S \in \mathcal{M}((x-1)^d)$ and the characteristic polynomial $(x-1)^d$ to determine $2^{v+1} - d$ further bits without changing the linear complexity. We can then interpret S as a sequence of $\mathcal{M}((x-1)^{2^{v+1}})$ and determine $L(S)$ with the Games-Chan algorithm. But the determination of the further bits can be omitted:

The sequence $S \in \mathcal{M}((x-1)^d)$ is in the subset $\mathcal{M}((x-1)^{2^v})$ of $\mathcal{M}((x-1)^d)$ if and only if the first d bits are the first bits of a sequence with period 2^v. If this is the case then $L(S)$ is obtained with the Games-Chan algorithm applied to the first 2^v bits of S.

Else by the Games-Chan algorithm $L = 2^v + L(S_1)$ where S_1 is the sequence in $\mathcal{M}((x-1)^{d-2^v}) \subset \mathcal{M}((x-1)^{2^v})$ (2^v-periodic) with first $d_1 = d - 2^v$ bits given by $s_0 + s_{2^v}, s_1 + s_{2^v+1}, \ldots, s_{d_1-1} + s_{2^v+d_1-1}$. We note that $S_1 \in \mathcal{M}((x-1)^{d-2^v})$ is completely determined by these d_1 bits. If d_1 is a power of 2 then one obtains $L(S_1)$ directly with the Games-Chan algorithm, else one continues recursively with the same argument as above. This yields the following obvious generalization of the Games-Chan algorithm for sequences in $\mathcal{M}((x-1)^d)$. We will use the notation $GC(S)$ for the output $L(S)$ of the Games-Chan algorithm applied to the 2^v-periodic binary sequence S.

Algorithm 0:
Initial values: $S = (s_0, s_1, \ldots, s_{d-1})$, d, $L = 0$.
Output: L, the linear complexity of the sequence $S \in \mathcal{M}((x-1)^d)$.

```
while $d \geq 1$ do
  $v = \lfloor \log_2 d \rfloor$; $d = d - 2^v$;
    if $d = 0$
      $L = L + GC(S)$;
    else
      $A = (s_0 + s_{2^v}, s_1 + s_{2^v+1}, \ldots, s_{d-1} + s_{2^v+d-1})$;
      $S = (s_0, s_1, \ldots, s_{2^v-1})$;
      if $A = (0, 0, \ldots, 0)$
      $L = L + GC(S)$;
    else
      $L = L + 2^v$; $S = A$;
    end if
  end if
end while
```

For another technique for the same purpose that adapts the more general Stamp-Martin algorithm [19] we refer to [18, Theorem 3.2].

3 Linear complexity for binary sequences in $\mathcal{M}((x^2+x+1)^{2^v})$

Let $S = s_0, s_1, s_2 \dots$ be a binary sequence in $\mathcal{M}((x^2+x+1)^{2^v})$, then with (1) and the observation that $f^*(x) = f(x)$ for $f(x) = (x^2+x+1)^{2^v}$ we obtain that $S(x) = g(x)/(x^2+x+1)^{2^v}$ for a unique polynomial $g(x) \in \mathbb{F}_2[x]$ of degree at most $2^{v+1}-1$. With equation (2) the linear complexity $L(S)$ of S is then given by

$$L(S) = 2^{v+1} - \deg(\gcd(g(x), (x^2+x+1)^{2^v})).$$

Since x^2+x+1 is irreducible over $\mathbb{F}_2$ the linear complexity is determined by the largest power of x^2+x+1 by which $g(x)$ can be divided. We note that $L(S) = 2j$ for an integer $0 \leq j \leq 2^v$.

We start with the description of the algorithm for determining the linear complexity of a sequence $S \in \mathcal{M}((x^2+x+1)^{2^v})$, and prove its correctness after giving an example.

Suppose that $S = (s_0, s_1, \dots, s_{2^{v+1}-1}) \in \mathcal{M}((x^2+x+1)^{2^v})$, then the linear complexity of S can be evaluated with the following procedure. With the same arguments as for the Games-Chan algorithm it is seen that also this procedure evaluates the linear complexity in linear time.

Algorithm 1:
Initial values: $S = (s_0, s_1, \dots, s_{2^{v+1}-1})$, $l = 2^{v+1}$, $L = 0$.
Output: L, the linear complexity of the sequence $S \in \mathcal{M}((x^2+x+1)^{2^v})$.

```
while $l > 2$ do
  $A = (s_{l/2}, s_{l/2+1}, \dots, s_{l-1})$;
  $B = (s_0 + s_{l/4} + s_{l/2}, s_1 + s_{1+l/4} + s_{1+l/2}, \dots, s_{l/2-1} + s_{3l/4-1} + s_{l-1})$;
    if $B = (0, 0, \dots, 0)$
      $S = A$; $l = l/2$;
    else
      $S = B$; $l = l/2$; $L = L + l$;
    end if
end while
if $S \neq (0, 0)$
  $L = L + l$;
```

end if

Example 1 $S = (1000110000100011)$, $l = 16$, $L = 0$.
According to the algorithm the string B is determined by adding componentwise modulo 2 the first half of the period, the centre part, and the second half of the period (i.e. string A). The initial string for the next step is either A or B depending whether B is the zero string or not.

Step 1:

$$\begin{array}{r|cccccccc} & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ A: & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ \hline B: & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \end{array}$$

$S = B, l = 8, L = 8$

Step 2:

$$\begin{array}{r|cccc} & 0 & 1 & 1 & 0 \\ & 1 & 0 & 1 & 1 \\ A: & 1 & 1 & 0 & 1 \\ \hline B: & 0 & 0 & 0 & 0 \end{array}$$

$S = A, l = 4, L = 8$

Step 3:

$$\begin{array}{r|cc} & 1 & 1 \\ & 1 & 0 \\ A: & 0 & 1 \\ \hline B: & 0 & 0 \end{array}$$

$S = A, l = 2$
$L = 8$

Final Step: $S = (01) \neq (00)$, *consequently* $L = 8 + 2 = 10$.

For proving the correctness of the algorithm we need some lemmas.

Lemma 1 *Let* $S = (s_0, s_1, \dots, s_{2^{v+1}-1})$ *be a sequence in* $\mathcal{M}((x^2 + x + 1)^{2^v})$*, then the polynomial* $g(x) = \sum_{i=0}^{2^{v+1}-1} a_i x^i$ *with* $S(x) = g(x)/(x^2 + x + 1)^{2^v}$ *is determined by*

$$a_i = \left\{ \begin{array}{rcl} s_i & : & 0 \leq i \leq 2^v - 1 \\ s_i + s_{i-2^v} & : & 2^v \leq i \leq 2^{v+1} - 1 \end{array} \right. . \tag{3}$$

Proof. Clearly the coefficients of $g(x)$ coincide with the first 2^{v+1} coefficients of $(x^2 + x + 1)^{2^v} S(x)$. Using $(x^2 + x + 1)^{2^v} = x^{2^{v+1}} + x^{2^v} + 1$ the lemma follows. □

Lemma 2 *Let* $g(x) = \sum_{i=0}^{2^{v+1}-1} a_i x^i \in \mathbb{F}_2[x]$ *be a polynomial of degree at most* $2^{v+1} - 1$ *for an integer* $v \geq 1$*, let* $q(x) = \sum_{i=0}^{2^v-1} q_i x^i$ *be the quotient and* $r(x) = \sum_{i=0}^{2^v-1} r_i x^i$ *be the remainder of* $g(x)$ *at division by* $(x^2 + x + 1)^{2^{v-1}}$*, i.e.* $g(x) = q(x)(x^2 + x + 1)^{2^{v-1}} + r(x)$*. Then*

$$\begin{aligned} q_i &= \left\{ \begin{array}{rcl} a_{i+2^v} + a_{i+3\cdot 2^{v-1}} & : & 0 \leq i \leq 2^{v-1} - 1 \\ a_{i+2^v} & : & 2^{v-1} \leq i \leq 2^v - 1 \end{array} \right. , \\ r_i &= \left\{ \begin{array}{rcl} a_i + a_{i+2^v} + a_{i+3\cdot 2^{v-1}} & : & 0 \leq i \leq 2^{v-1} - 1 \\ a_i + a_{i+2^{v-1}} & : & 2^{v-1} \leq i \leq 2^v - 1 \end{array} \right. . \end{aligned} \tag{4}$$

Proof. The proof easily follows by comparing the coefficients a_i of x^i in $g(x)$ with the coefficients of x^i in $q(x)(x^2 + x + 1)^{2^{v-1}} + r(x)$, where one has to distinguish between the cases $0 \leq i < 2^{v-1}$, $2^{v-1} \leq i < 2^v$, $2^v \leq i < 3 \cdot 2^{v-1}$ and $3 \cdot 2^{v-1} \leq i < 2^{v+1}$. □

Remark 1 *Let $S \in \mathcal{M}((x^2+x+1)^{2^v})$ and let $g(x)$ be the polynomial of degree less than 2^{v+1} for which we have $S(x) = g(x)/(x^2+x+1)^{2^v}$. Then with* (3) *and* (4) *we obtain*

$$r_i = \begin{cases} s_{i+2^{v-1}} + s_{i+2^v} + s_{i+3\cdot 2^{v-1}} & : \quad 0 \le i \le 2^{v-1}-1 \\ s_{i-2^{v-1}} + s_i + s_{i+2^{v-1}} & : \quad 2^{v-1} \le i \le 2^v - 1 \end{cases} \tag{5}$$

for the remainder $r(x)$ defined in Lemma 2. As easily seen, in the important case that $r(x) = 0$ the quotient $q(x)$ defined as in Lemma 2 is determined by

$$q_i = \begin{cases} s_{i+3\cdot 2^{v-1}} & : \quad 0 \le i \le 2^{v-1}-1 \\ s_{i+2^{v-1}} & : \quad 2^{v-1} \le i \le 2^v - 1 \end{cases}.$$

Before we show the correctness of Algorithm 1 we outline a general procedure based on (2) to obtain the linear complexity of a binary sequence $S \in \mathcal{M}(f^{2^v})$ if $f(x)$ is an irreducible polynomial of degree d in $\mathbb{F}_2[x]$.

Let $g(x)$ be the unique polynomial of degree at most $d2^v - 1$ with $S(x) = g(x)/f^*(x)^{2^v}$, and suppose that

$$g(x) = q(x) f^*(x)^{2^{v-1}} + r(x), \ \deg(r) < d2^{v-1}.$$

Clearly if $r(x) = 0$, then

$$S(x) = q(x)/f^*(x)^{2^{v-1}},$$

and consequently $S \in \mathcal{M}(f^{2^{v-1}})$.

If $r(x) \neq 0$ then

$$\gcd(g(x), f^*(x)^{2^v}) = \gcd(r(x), f^*(x)^{2^{v-1}}),$$

and thus with equation (2) we obtain $L(S) = d2^{v-1} + L(S_1)$ with $S_1(x) = r(x)/f^*(x)^{2^{v-1}}$. This yields a procedure in v steps (plus a final step) for the evaluation of $L(S)$. Putting $g(x) = r_0(x)$, step m, $1 \le m \le v$, is performed as follows:

Determine $q_m(x)$, $r_m(x)$ with $r_{m-1}(x) = q_m(x) f^*(x)^{2^{v-m}} + r_m(x)$, $\deg(r_m) < d2^{v-m}$.

If $r_m(x) = 0$ then $L(S_{m-1}) = L(S_m)$ with $S_m(x) = q_m(x)/f^*(x)^{2^{v-m}}$. Put $r_m = q_m$.

Else $L(S_{m-1}) = d2^{v-m} + L(S_m)$ with $S_m(x) = r_m(x)/f^*(x)^{2^{v-m}}$.

Theorem 1 *Algorithm 1 is correct.*

Proof. We show that Algorithm 1 accomplishes the above described procedure for $f(x) = x^2 + x + 1$.

Clearly, Algorithm 1 operates in v steps plus the final step. We fix the following notation: $A_m(S)$ and $B_m(S)$ shall be the strings A and B, respectively, in step m of Algorithm 1. Evidently each of these strings has length 2^{v+1-m}. The polynomial $r_m(x)$ shall be defined as above, its coefficients shall be denoted by $r_i^{(m)}$, i.e.

$$r_m(x) = r_0^{(m)} + r_1^{(m)}x + \cdots + r_{2^{v+1-m}-1}^{(m)} x^{2^{v+1-m}-1}.$$

As above S_m denotes the sequence with generating function $S_m(x) = r_m/(x^2 + x + 1)^{2^{v-m}}$, its terms shall be denoted by $s_i^{(m)}$.

Let $S = (s_0, s_1, \ldots, s_{2^{v+1}-1})$. By equation (5) the string $B_1(S)$ calculated as described in Algorithm 1 exactly contains the coefficients of $r_1(x)$, namely

$$B_1(S) = (r_{2^{v-1}}^{(1)}, \ldots, r_{2^v-1}^{(1)}, r_0^{(1)}, \ldots, r_{2^{v-1}-1}^{(1)}). \tag{6}$$

Consequently if $B_1(S) \neq (0, 0, \ldots, 0) := \mathbf{0}$, then $L(S) = 2^v + L(S_1)$.

In the next step instead of using the initial terms of the sequence S_1 as input for the calculation of $B_2(S)$ we directly use the string (6). We claim that then

$$B_2(S) = (s_0^{(2)}, s_1^{(2)}, \ldots, s_{2^{v-1}-1}^{(2)}).$$

This claim will be shown (for general m) below at the end of the proof. Clearly again $B_2(S) \neq \mathbf{0}$ implies $r_2(x) \neq 0$. As required, Algorithm 1 then increases the value L by 2^{v-1}. The string $B_3(S)$ for the next step is then calculated with the terms of the sequence S_2, hence $B_3(S)$ exactly contains the coefficients of the next remainder $r_3(x)$. Certainly the same argument works at any step m of Algorithm 1.

Now we consider the case that in a step m, $1 \leq m \leq v$, we have $B_m(S) = \mathbf{0}$. Then $r_m(x) = 0$ and $S_{m-1} = S_m \in \mathcal{M}((x^2+x+1)^{2^{v-m}})$. We have to distinguish two cases.

If we obtained $B_m(S)$ from the coefficients of $r_{m-1}(x)$ then $A_m(S) = (r_0^{(m-1)}, r_1^{(m-1)}, \ldots, r_{2^{v+1-m}-1}^{(m-1)})$ which by (3) coincides with the string $(s_0^{(m-1)}, s_1^{(m-1)}, \ldots, s_{2^{v+1-m}-1}^{(m-1)})$ of the first 2^{v+1-m} terms of the sequence $S_m = S_{m-1}$. Since $S_m \in \mathcal{M}((x^2 + x + 1)^{2^{v-m}})$, we arrive at an initial situation for Algorithm 1, where the first terms of the sequence whose linear complexity has to be determined are the input for the algorithm.

If we obtained $B_m(S)$ directly from the terms of the sequence, then $A_m(S) = (s_{2^{v+1-m}}^{(m-1)}, s_{2^{v+1-m}+1}^{(m-1)}, \ldots, s_{2^{v+2-m}-1}^{(m-1)})$. Taking the string $A_m(S)$ as starting values we obtain a shifted version of $S_{m-1} = S_m \in \mathcal{M}((x^2 + x + 1)^{2^{v-m}})$, which of course has the same linear complexity. Consequently we again arrive at an initial situation for Algorithm 1 to determine $L(S_m)$.

It remains to prove the correctness of the claim stated above.

Suppose at step m we obtained , $B_m(S) = (r_{2^{v-m}}^{(m)}, \ldots, r_{2^{v+1-m}-1}^{(m)}, r_0^{(m)}, \ldots, r_{2^{v-m}-1}^{(m)}) \neq \mathbf{0}$. If we denote the ith term of the string $B_{m+1}(S)$ by b_i, $0 \leq i \leq 2^{v-m} - 1$, then

$$b_i = \begin{cases} r_i^{(m)} + r_{i+2^{v-m}}^{(m)} + r_{i+3\cdot 2^{v-m-1}}^{(m)} & : \quad 0 \leq i \leq 2^{v-m-1} - 1 \\ r_i^{(m)} + r_{i-2^{v-m-1}}^{(m)} + r_{i+2^{v-m}}^{(m)} & : \quad 2^{v-m-1} \leq i \leq 2^{v-m} - 1 \end{cases}.$$

We have to show that $b_i = s_i^{(m+1)}$, $0 \le i \le 2^{v-m}-1$. Since $s_i^{(m+1)}$ are the terms of the sequence with the generating function $S_{m+1} = r_{m+1}(x)/(x^2+x+1)^{2^{v-m-1}}$ we first determine the coefficients of $r_{m+1}(x)$ from the coefficients of $r_m(x)$. By equation (5) we obtain

$$r_i^{(m+1)} = \begin{cases} r_i^{(m)} + r_{i+2^{v-m-1}}^{(m)} + r_{i+3\cdot 2^{v-m-2}}^{(m)} & : \quad 0 \le i \le 2^{v-m-2}-1 \\ r_i^{(m)} + r_{i+2^{v-m-2}}^{(m)} & : \quad 2^{v-m-2} \le i \le 2^{v-m-1}-1 \end{cases}.$$

Since by (3) we have $s_i^{(m+1)} = r_i^{(m+1)}$ for $0 \le i \le 2^{v-m-2}-1$, and $s_i^{(m+1)} = r_i^{(m+1)} + r_{i-2^{v-m-2}}^{(m+1)}$ for $2^{v-m-2} \le i \le 2^{v-m-1}-1$ we see that $b_i = s_i^{(m+1)}$ holds for $0 \le i \le 2^{v-m}-1$. □

We remark that similar algorithms can be established for binary sequences in $\mathcal{M}(f^{2^v})$ for other irreducible polynomials $f(x) \in \mathbb{F}_2[x]$ based on the procedure described before the proof of Theorem 1. One of the reasons of the particular simplicity and beauty of Algorithm 1 where $f(x) = x^2 + x + 1$ is the fact that we can use the initial terms of the sequence as input, and arrive at the linear complexity with recursively repeating the initial step without the necessity of transforming sequence terms into coefficients of corresponding polynomials. This simple structure of Algorithm 1 allows further analysis. In the next section we make use of this simple structure and present an algorithm for the calculation of the k-error linear complexity of binary sequences in $\mathcal{M}((x^2+x+1)^{2^v})$.

4 k-error linear complexity for binary sequences in $\mathcal{M}((x^2+x+1)^{2^v})$

First we have to give an interpretation for the k-error linear complexity of a sequence $S \in \mathcal{M}(f)$ for an arbitrary characteristic polynomial $f(x) \in \mathbb{F}_2[x]$: The k-error linear complexity of S is the smallest linear complexity of a sequence T in $\mathcal{M}(f)$ we can obtain by changing at most k terms in the initial string of S. This definition precisely coincides with the definition of the k-error linear complexity of the finite initial string z of S in the set $\mathcal{M}(f)$ given in the article [18].

We recall that $\mathcal{M}(f)$, $\deg(f) = d$, is a d-dimensional vector space (see [11, Chapter 8]) whose elements can naturally be identified with strings of length d. For each divisor $f_1(x)$ of $f(x)$, $\mathcal{M}(f_1)$ is a subspace of $\mathcal{M}(f)$ of dimension $d_1 = \deg(f_1)$. Hence we can interpret the k-error linear complexity of a sequence $S \in \mathcal{M}(f)$ as the lowest dimension of a subspace of $\mathcal{M}(f)$ that contains a sequence T such that the initial strings of S and T have Hamming distance at most k.

The basic idea for the algorithm to determine the k-error linear complexity of sequences in $\mathcal{M}((x^2+x+1)^{2^v})$ is as follows: Since $B_m(S) = \mathbf{0}$ in the mth step of Algorithm 1 prevents 2^{v+1-m} of being added to the value for the linear complexity and the total of all remaining possible additions is only 2^{v+1-m}, we have to convert $B_m(S)$ to the zero string if we can. This has to be made by appropriate bit changes in the original string without that results of previous steps are destroyed, and the total number of bit changes in the original string must not exceed k. For this purpose we observe that the sets of bits of B_{m-1} (A_{m-1}) from which we calculate b_i and $b_{i+2^{v-m}}$ of the string $B_m(S)$ are not distinct. One bit-change in B_{m-1} (A_{m-1}) may convert both bits, b_i and $b_{i+2^{v-m}}$. Consequently we consider the pair of bits b_i and $b_{i+2^{v-m}}$ of $B_m(S)$ $(A_m(S))$, and assign two 2-tuples (μ, δ) and (ν, δ) to b_i and $b_{i+2^{v-m}}$. We indicate this with the notation

$$(b_i)_{(\mu,\delta)} \quad (b_{i+2^{v-m}})_{(\nu,\delta)}.$$

The parameter μ is defined to be the minimal number of bit changes in the original string required in order to change b_i but not $b_{i+2^{v-m}}$, without adversely affecting any intermediate results. The parameter ν is defined to be the minimal number of bit changes in the original string required in order to change $b_{i+2^{v-m}}$ but not b_i, without adversely affecting any intermediate results, and δ is the minimal number of bit changes in the original string required in order to change both bits, b_i and $b_{i+2^{v-m}}$, without adversely affecting any intermediate results. Clearly by this definition we have to assign $(1, 2)$ to each bit in the initial string, the string containing the first 2^{v+1} bits of S.
It is evident how to use the cost-tuples to determine the total "costs" for converting the complete string $B_m(S)$ into the zero string. The crucial question is how to determine the cost-tuples in $B_m(S)$ and if necessary in $A_m(S)$ from the cost-tuples in step $m-1$.

From Algorithm 1 we can see that the pair $b_i, b_{i+2^{v-m}}$ of bits in $B_m(S)$ is determined from four bits from the previous step $m-1$. For simplicity we call these bits $\beta_1, \beta_2, \beta_3$ and β_4 and we let $b_i, b_{i+2^{v-m}}$ be calculated by

$$\begin{array}{cc} (\beta_1)_{(\mu_1,\delta_1)} & (\beta_2)_{(\mu_2,\delta_2)} \\ (\beta_2)_{(\mu_2,\delta_2)} & (\beta_3)_{(\nu_1,\delta_1)} \\ (\beta_3)_{(\nu_1,\delta_1)} & (\beta_4)_{(\nu_2,\delta_2)} \\ \hline b_i & b_{i+2^{v-m}}. \end{array} \tag{7}$$

We remark that the bits β_1, β_3 (β_2, β_4) from step $m-1$ are not independent. Accordingly, these bits share the second component δ_1 (δ_2) in their cost-tuples. If for instance β_1 shall be changed but not β_3, then μ_1 describes the costs, but if β_1 and β_3 shall both be changed, then the costs are given by δ_1 (compare the second and the fourth table in Case 00 given below).
From the following illustrations we can see the "costs" for all possible cases, case 00 where both bits $b_i, b_{i+2^{v-m}}$ are not changed, case 01 where b_i is not changed but $b_{i+2^{v-m}}$, i.e 1 is added to $b_{i+2^{v-m}}$ modulo 2, case 10 where b_i is changed but not $b_{i+2^{v-m}}$, and case 11 where the tuple (11) is added modulo 2 to $(b_i b_{i+2^{v-m}})$ component wise, i.e. both bits are changed. In the following tables 0 shall stand for "keeping" the value of the bit in the respective position in (7), and 1 means that the bit in the respective position in (7) is changed, i.e. 1 is added modulo 2 to the bit. In the case of a change we indicate the cost of the change with a subscript. For instance the second table of Case 00 describes the following situation: Firstly the bit β_1 in (7) is changed but β_3 is not changed,

for which the "costs" are given by μ_1 by definition. Secondly both bits, β_2 and β_4 in (7) are changed, which by definition costs δ_2. The sums of bits determined according to Algorithm 1 are then unchanged (Case 00).

Case 00: $(b_i b_{i+2^{v-m}}) \to (b_i b_{i+2^{v-m}})$

$$\begin{array}{ll} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \hline 0 & 0 \end{array} \qquad \begin{array}{ll} 1_{\mu_1} & 1_{\delta_2} \\ 1_{\delta_2} & 0 \\ 0 & 1_{\delta_2} \\ \hline 0 & 0 \end{array} \qquad \begin{array}{ll} 0 & 1_{\mu_2} \\ 1_{\mu_2} & 1_{\nu_1} \\ 1_{\nu_1} & 0 \\ \hline 0 & 0 \end{array} \qquad \begin{array}{ll} 1_{\delta_1} & 0 \\ 0 & 1_{\delta_1} \\ 1_{\delta_1} & 1_{\nu_2} \\ \hline 0 & 0 \end{array}$$

Case 01: $(b_i b_{i+2^{v-m}}) \to (b_i b_{i+2^{v-m}}) + (01)$

$$\begin{array}{ll} 1_{\mu_1} & 1_{\mu_2} \\ 1_{\mu_2} & 0 \\ 0 & 0 \\ \hline 0 & 1 \end{array} \qquad \begin{array}{ll} 0 & 0 \\ 0 & 0 \\ 0 & 1_{\nu_2} \\ \hline 0 & 1 \end{array} \qquad \begin{array}{ll} 1_{\delta_1} & 0 \\ 0 & 1_{\delta_1} \\ 1_{\delta_1} & 0 \\ \hline 0 & 1 \end{array} \qquad \begin{array}{ll} 0 & 1_{\delta_2} \\ 1_{\delta_2} & 1_{\nu_1} \\ 1_{\nu_1} & 1_{\delta_2} \\ \hline 0 & 1 \end{array}$$

Case 10: $(b_i b_{i+2^{v-m}}) \to (b_i b_{i+2^{v-m}}) + (10)$

$$\begin{array}{ll} 1_{\mu_1} & 0 \\ 0 & 0 \\ 0 & 0 \\ \hline 1 & 0 \end{array} \qquad \begin{array}{ll} 0 & 1_{\delta_2} \\ 1_{\delta_2} & 0 \\ 0 & 1_{\delta_2} \\ \hline 1 & 0 \end{array} \qquad \begin{array}{ll} 1_{\delta_1} & 1_{\mu_2} \\ 1_{\mu_2} & 1_{\delta_1} \\ 1_{\delta_1} & 0 \\ \hline 1 & 0 \end{array} \qquad \begin{array}{ll} 0 & 0 \\ 0 & 1_{\nu_1} \\ 1_{\nu_1} & 1_{\nu_2} \\ \hline 1 & 0 \end{array}$$

Case 11: $(b_i b_{i+2^{v-m}}) \to (b_i b_{i+2^{v-m}}) + (11)$:

$$\begin{array}{ll} 0 & 1_{\mu_2} \\ 1_{\mu_2} & 0 \\ 0 & 0 \\ \hline 1 & 1 \end{array} \qquad \begin{array}{ll} 1_{\mu_1} & 0 \\ 0 & 0 \\ 0 & 1_{\nu_2} \\ \hline 1 & 1 \end{array} \qquad \begin{array}{ll} 0 & 0 \\ 0 & 1_{\nu_1} \\ 1_{\nu_1} & 0 \\ \hline 1 & 1 \end{array} \qquad \begin{array}{ll} 1_{\delta_1} & 1_{\delta_2} \\ 1_{\delta_2} & 1_{\delta_1} \\ 1_{\delta_1} & 1_{\delta_2} \\ \hline 1 & 1 \end{array}$$

We point out that in each case there are exactly four possibilities of changing $(b_i b_{i+2^{v-m}})$, one case which leaves the bits in the last of the three rows unchanged, one case each for changing exactly one of these two bits and one case in which both of these bits are changed. This is of importance since these bits precisely form the bit pairs in the string $A_m(S)$. We choose the letters $\mathcal{K}$ and $\mathcal{C}$ to indicate keeping and changing bits in the third row, and refer to the possible cases with the notation $\mathcal{KK}$, $\mathcal{KC}$, $\mathcal{CK}$ and $\mathcal{CC}$.

The cost for each of the cases can easily be seen from the illustrations above. For instance the third table in case 10 says that the "costs" for changing $(b_i b_{i+2^{v-m}})$ into $(b_i b_{i+2^{v-m}}) + (10)$ such that additionally the first bit in the third row is changed and the second one is unchanged (case $\mathcal{CK}$) is given by $\mu_2 + \delta_1$. For this value we will use the notation $cost(\mathcal{CK})$. (Strictly speaking one should write $cost(\mathcal{CK})(10)$ for $cost(\mathcal{CK})$ in the case 10, but we will omit this distinction since it will not be necessary at describing the algorithm.)

We will mostly be interested into converting $(b_i b_{i+2^{v-m}})$ into (00). If for instance $(b_i b_{i+2^{v-m}}) = (01)$ we evidently have to choose one of the options in case

01. The minimal costs for changing $(b_i b_{i+2^{v-m}})$ into (00) will be denoted by $cost[i]$. The following table summarizes these ”costs” for all 16 possible cases.

	$\mathcal{K}\mathcal{K}$	$\mathcal{K}\mathcal{C}$	$\mathcal{C}\mathcal{K}$	$\mathcal{C}\mathcal{C}$	$cost[i]$
00	0	$\mu_1 + \delta_2$	$\mu_2 + \nu_1$	$\nu_2 + \delta_1$	0
01	$\mu_1 + \mu_2$	ν_2	δ_1	$\nu_1 + \delta_2$	$\min(\mu_1 + \mu_2, \nu_2, \delta_1, \nu_1 + \delta_2)$
10	μ_1	δ_2	$\mu_2 + \delta_1$	$\nu_1 + \nu_2$	$\min(\mu_1, \delta_2, \mu_2 + \delta_1, \nu_1 + \nu_2)$
11	μ_2	$\mu_1 + \nu_2$	ν_1	$\delta_1 + \delta_2$	$\min(\mu_2, \mu_1 + \nu_2, \nu_1, \delta_1 + \delta_2)$

By convention at calculating the minimum we take the first number in the list, if two or more are equal.
Obviously the total costs for converting the complete string $B_m(S)$ into the zero string are then given by $cost = \sum_{i=0}^{2^{v-m}-1} cost[i]$. If $cost$ does not exceed the number of allowed bit changes in step m, then $B_m(S)$ is converted into the zero string and according to Algorithm 1 we use $A_m(S)$ as input for the next step. We note that some bits of $A_m(S)$ have been changed in order to produce $B_m(S) = \mathbf{0}$.

We have the necessity of assigning appropriate cost-tuples (μ, δ), (ν, δ) to each bit pair $a_i a_{i+2^{v-m}}$, $0 \leq i \leq 2^{v-m} - 1$, in $A_m(S)$.
Let us consider the case that $cost[i] = cost(\mathcal{K}\mathcal{C})$, and thus we used option $\mathcal{K}\mathcal{C}$ for converting $(b_i b_{i+2^{v-m}})$ into (00). If we now want to alter bit a_i and keep bit $a_{i+2^{v-m}}$ we have to change our strategy and use option $\mathcal{C}\mathcal{C}$ to convert $(b_i b_{i+2^{v-m}})$ into (00). The cost μ for changing a_i and keeping $a_{i+2^{v-m}}$ in the next step is thus given by $cost(\mathcal{C}\mathcal{C}) - cost[i]$. Similarly we obtain $\nu = cost(\mathcal{K}\mathcal{K}) - cost[i]$, $\delta = cost(\mathcal{C}\mathcal{K}) - cost[i]$ in the case that $cost[i] = cost(\mathcal{K}\mathcal{C})$. The parameters μ, ν, δ for the other cases, i.e. the cases $cost[i] = cost(\mathcal{K}\mathcal{K})$, $cost[i] = cost(\mathcal{C}\mathcal{K})$ and $cost[i] = cost(\mathcal{C}\mathcal{C})$, are obtained analogously and are collected in the following table.

$cost[i]$	μ	ν	δ
$cost(\mathcal{K}\mathcal{K})$	$cost(\mathcal{C}\mathcal{K}) - cost[i]$	$cost(\mathcal{K}\mathcal{C}) - cost[i]$	$cost(\mathcal{C}\mathcal{C}) - cost[i]$
$cost(\mathcal{K}\mathcal{C})$	$cost(\mathcal{C}\mathcal{C}) - cost[i]$	$cost(\mathcal{K}\mathcal{K}) - cost[i]$	$cost(\mathcal{C}\mathcal{K}) - cost[i]$
$cost(\mathcal{C}\mathcal{K})$	$cost(\mathcal{K}\mathcal{K}) - cost[i]$	$cost(\mathcal{C}\mathcal{C}) - cost[i]$	$cost(\mathcal{K}\mathcal{C}) - cost[i]$
$cost(\mathcal{C}\mathcal{C})$	$cost(\mathcal{K}\mathcal{C}) - cost[i]$	$cost(\mathcal{C}\mathcal{K}) - cost[i]$	$cost(\mathcal{K}\mathcal{K}) - cost[i]$

Combining the two tables above yields the subsequent procedure for the determination of pairs of bits of the (adapted) string $A_m(S)$ and their cost-tuples (μ, δ), (ν, δ).

Procedure A$(\mu_1, \nu_1, \delta_1, \mu_2, \nu_2, \delta_2, \beta_3, \beta_4, b_i, b_{i+l/2})$
Input: bits $\beta_3, \beta_4, b_i, b_{i+l/2}$, cost-integers $\mu_1, \nu_1, \delta_1, \mu_2, \nu_2, \delta_2$.
Output: $a_i, a_{i+l/2}$, a pair of bits in an (adapted) string $A(S)$, the corresponding costs $(\mu, \nu, \delta) := (\mu, \nu, \delta)_{i_A}$.

```
if $b_i b_{i+l/2} = 00$
  $cost(\mathcal{K}\mathcal{K}) = 0$; $cost(\mathcal{K}\mathcal{C}) = \mu_1 + \delta_2$; $cost(\mathcal{C}\mathcal{K}) = \mu_2 + \nu_1$; $cost(\mathcal{C}\mathcal{C}) = \nu_2 + \delta_1$;
else
  if $b_i b_{i+l/2} = 01$
```

```
        $cost(\mathcal{KK}) = \mu_1 + \mu_2$; $cost(\mathcal{KC}) = \nu_2$; $cost(\mathcal{CK}) = \delta_1$; $cost(\mathcal{CC}) = \nu_1 + \delta_2$;
    else
        if $b_i b_{i+l/2} = 10$
            $cost(\mathcal{KK}) = \mu_1$; $cost(\mathcal{KC}) = \delta_2$; $cost(\mathcal{CK}) = \mu_2 + \delta_1$;
            $cost(\mathcal{CC}) = \nu_1 + \nu_2$;
        else
            $cost(\mathcal{KK}) = \mu_2$; $cost(\mathcal{KC}) = \mu_1 + \nu_2$; $cost(\mathcal{CK}) = \nu_1$;
            $cost(\mathcal{CC}) = \delta_1 + \delta_2$;
        end if
    end if
end if
$cost[i] = min(cost(\mathcal{KK}), cost(\mathcal{KC}), cost(\mathcal{CK}), cost(\mathcal{CC}))$;
if $cost[i] = cost(\mathcal{KK})$
    $\mu = cost(\mathcal{CK}) - cost[i]$; $\nu = cost(\mathcal{KC}) - cost[i]$; $\delta = cost(\mathcal{CC}) - cost[i]$;
    $a_i = \beta_3$; $a_{i+l/2} = \beta_4$;
else
    if $cost[i] = cost(\mathcal{KC})$
        $\mu = cost(\mathcal{CC}) - cost[i]$; $\nu = cost(\mathcal{KK}) - cost[i]$; $\delta = cost(\mathcal{CK}) - cost[i]$;
        $a_i = \beta_3$; $a_{i+l/2} = \beta_4 + 1 \bmod 2$;
    else
        if $cost[i] = cost(\mathcal{CK})$
            $\mu = cost(\mathcal{KK}) - cost[i]$; $\nu = cost(\mathcal{CC}) - cost[i]$; $\delta = cost(\mathcal{KC}) - cost[i]$;
            $a_i = \beta_3 + 1 \bmod 2$; $a_{i+l/2} = \beta_4$;
        else
            $\mu = cost(\mathcal{KC}) - cost[i]$; $\nu = cost(\mathcal{CK}) - cost[i]$; $\delta = cost(\mathcal{KK}) - cost[i]$;
            $a_i = \beta_3 + 1 \bmod 2$; $a_{i+l/2} = \beta_4 + 1 \bmod 2$;
        end if
    end if
end if
```

If *cost* exceeds the number of allowed bit changes in step m, then we cannot convert $B_m(S)$ into the zero string and according to Algorithm 1 we use $B_m(S)$ as input for the next step. Thus we have to assign appropriate cost-tuples (μ, δ), (ν, δ) to each bit pair $b_i b_{i+2^{v-m}}$, $0 \le i \le 2^{v-m} - 1$, in $B_m(S)$. The costs μ for changing b_i but not $b_{i+2^{v-m}}$ are obtained from the tables in case 10. Thus $\mu = \min(\mu_1, \delta_2, \mu_2 + \delta_1, \nu_1 + \nu_2)$, and similarly we see that $\nu = \min(\mu_1 + \mu_2, \nu_2, \delta_1, \nu_1 + \delta_2)$ and $\delta = \min(\mu_2, \mu_1 + \nu_2, \nu_1, \delta_1 + \delta_2)$. With these considerations we obtain the following algorithm to determine the k-error linear complexity of binary sequences S in $\mathcal{M}((x^2 + x + 1)^{2^v})$.

Algorithm 2:
Initial values: $S = (s_0, s_1, \ldots, s_{2^{v+1}-1})$, $l = 2^{v+1}$, $L = 0$, k, and $(\mu, \nu, \delta)_i = (1, 1, 2)$ for $i = 0, 1, \ldots, l - 1$.
Output: L, the k-error linear complexity of the sequence $S \in \mathcal{M}((x^2+x+1)^{2^v})$.

```
while $l > 2$ do
```

```
    $a_i = s_{l/2+i}$, $b_i = s_i + s_{l/4+i} + s_{l/2+i}$ for $i = 0, 1, \ldots, l/2 - 1$;
    $B = (b_0, b_1, \ldots, b_{l/2-1})$;
    for $i = 0, 1 \ldots, l/2 - 1$ do
        $(\mu_1, \nu_1, \delta_1) = (\mu, \nu, \delta)_i; (\mu_2, \nu_2, \delta_2) = (\mu, \nu, \delta)_{i+l/2}$;
        $\mu_B = \min(\mu_1, \delta_2, \mu_2 + \delta_1, \nu_1 + \nu_2)$; $\nu_B = \min(\mu_1 + \mu_2, \nu_2, \delta_1, \nu_1 + \delta_2)$;
        $\delta_B = \min(\mu_2, \mu_1 + \nu_2, \nu_1, \delta_1 + \delta_2)$; $(\mu, \nu, \delta)_{i_B} = (\mu_B, \nu_B, \delta_B)$;
        $cost[i] = \begin{cases} 0 & : \quad (b_i b_{i+l/2}) = (00) \\ \nu_B & : \quad (b_i b_{i+l/2}) = (01) \\ \mu_B & : \quad (b_i b_{i+l/2}) = (10) \\ \delta_B & : \quad (b_i b_{i+l/2}) = (11) \end{cases}$;
    $T = \sum_{i=0}^{l/2-1} cost[i]$;
    if $T \geq k$
        $L = L + l/2$; $l = l/2$; $S = B$;
        $(\mu, \nu, \delta)_i = (\mu, \nu, \delta)_{i_B}$, for $i = 0, 1, \ldots, l - 1$;
    else
        for $i = 0, 1 \ldots, l/2 - 1$ do
            $(\mu_1, \nu_1, \delta_1) = (\mu, \nu, \delta)_i; (\mu_2, \nu_2, \delta_2) = (\mu, \nu, \delta)_{i+l/2}$;
            $\beta_3 = a_i$; $\beta_4 = a_{i+l/4}$;
            Procedure A$(\mu_1, \nu_1, \delta_1, \mu_2, \nu_2, \delta_2, \beta_3, \beta_4, b_i, b_{i+l/2})$;
        $k = k - T$; $l = l/2$; $S = (a_0, a_1, \ldots, a_{l-1})$;
        $(\mu, \nu, \delta)_i = (\mu, \nu, \delta)_{i_A}$, for $i = 0, 1, \ldots, l - 1$;
end while
if $S = (01)$ and $\nu > k$ or $S = (10)$ and $\mu > k$ or $S = (11)$ and $\delta > k$
    $L = L + l$;
end if
```

Next we analyse the time complexity of Algorithm 2:
The expensive parts in Algorithm 2 are the calculation of T, the calculation of the integers giving the "costs", and Procedure A. We first remark that at each step of the algorithm the values in the cost tuples can at most double. Consequently in the step of Algorithm 2 when l takes the value 2^{v+1-m} (step m) every integer in the cost tuples has bit length at most $m + 2$. Hence in step m the calculation of T requires at most $(m + 2)2^{v-m}$ bit operations.
For each $i = 0, 1, \ldots, l/2 - 1 = 2^{v-m} - 1$ (step m) in the first "for" loop in Algorithm 2, the calculation of the cost integers μ_B, ν_B, δ_B is accomplished by a constant number of additions and comparisons of $(m+2)$-bit integers (step m), and thus $c_1(m+2)2^{v-m}$ bit operations for a constant c_1. In the mth step of the algorithm, Procedure A is performed (if at all) $l/2 = 2^{v-m}$ times. Procedure A consists of a constant number of operations with $(m+2)$-bit integers, which again gives $c_2(m + 2)2^{v-m}$ bit operations for a constant c_2. Using this observations and the fact that for constants c, b we have

$$c \sum_{m=0}^{v} (m + b)2^{v-m} = K_{c,b}2^{v+1} + \lambda(v),$$

where $K_{c,b}$ is a constant only depending on c, b, and λ is a linear polynomial in

v, we see that Algorithm 2 is a linear time algorithm.

In [10], Lauder and Paterson show how to transform the algorithm in [19] that determines the k-error linear complexity of a sequence $S \in \mathcal{M}(x^{2^v} - 1)$ for a fixed value of k, into an efficient algorithm that determines the k-error linear complexity for all values of k simultaneously. We can adapt Algorithm 2 similarly to an algorithm determining the k-error linear complexity for all values of k simultaneously. Here we follow precisely the approach in [10], thus we may refer to [10] for details and only present an example for Algorithm 2, respectively its adaption to an algorithm for determining the k-error linear complexity for all k simultaneously.

We first recall some definitions and notations from [10]. The *k-error linear complexity spectrum $ELCS(S)$* of a binary sequence S is the ordered list of the k-error linear complexities of S, i.e. $L_0(S) = L(S), L_1(S), \ldots, L_{wt(S)}(S) = 0$ where $wt(S)$ denotes the number of nonzero elements in the initial string of S. Clearly $ELCS(S)$ is given by all points $(k, L_k(S))$ for which we have $L_{k'}(S) > L_k(S)$ for all $k' < k$. We call these points *critical points* (CP), include $(0, L(S))$, and call the list of these points the *critical error linear complexity spectrum* $CELCS(S)$. In the example below we determine the $CELCS(S)$ of a given sequence $S \in \mathcal{M}((x^2+x+1)^{2^v})$. The procedure explores parts of a binary tree since in every step of Algorithm 2 the following step is either performed with the sequence corresponding to the string A or with the sequence corresponding to the string B. At each depth of the tree - following the notation in [10] -, the parameter tsf in $CELCS(S, tsf, lim, L)$ refers to the *total* costs of changes made to the sequence *so far*, and the variable lim marks a *limit* to the total costs of changes one should consider when searching the corresponding part of the tree. The last variable L is the current value for L (which at the end of the algorithm gives the linear complexity) at that part of the binary tree. In the example below, T denotes the costs for converting B into the zero string.

Example 2 $S = (0110110011100011)$.

$$
\begin{array}{rcccccccc}
 & 0_{(1,2)} & 1_{(1,2)} & 1_{(1,2)} & 0_{(1,2)} & 1_{(1,2)} & 1_{(1,2)} & 0_{(1,2)} & 0_{(1,2)} \\
 & 1_{(1,2)} & 1_{(1,2)} & 0_{(1,2)} & 0_{(1,2)} & 1_{(1,2)} & 1_{(1,2)} & 1_{(1,2)} & 0_{(1,2)} \\
 & 1_{(1,2)} & 1_{(1,2)} & 1_{(1,2)} & 0_{(1,2)} & 0_{(1,2)} & 0_{(1,2)} & 1_{(1,2)} & 1_{(1,2)} \\
\cline{2-9}
B: & 0_{(1,1)} & 1_{(1,1)} & 0_{(1,1)} & 0_{(1,1)} & 0_{(1,1)} & 0_{(1,1)} & 0_{(1,1)} & 1_{(1,1)} \\
A: & 1_{(2,3)} & 1_{(2,1)} & 1_{(2,3)} & 0_{(2,1)} & 0_{(3,3)} & 0_{(1,1)} & 1_{(3,3)} & 0_{(1,1)} \\
T: & 2 & & & & & & &
\end{array}
$$

$CELCS(B, 0, 1, 8)$

$$
\begin{array}{rcccc}
 & 0_{(1,1)} & 1_{(1,1)} & 0_{(1,1)} & 0_{(1,1)} \\
 & 0_{(1,1)} & 0_{(1,1)} & 0_{(1,1)} & 0_{(1,1)} \\
 & 0_{(1,1)} & 0_{(1,1)} & 0_{(1,1)} & 1_{(1,1)} \\
\cline{2-5}
B: & 0_{(1,1)} & 1_{(1,1)} & 0_{(1,1)} & 0_{(1,1)} \\
A: & 0_{(2,2)} & 1_{(1,2)} & 0_{(2,2)} & 0_{(2,2)} \\
T: & 1 & & &
\end{array}
$$

$CELCS(A, 2, 16, 0)$

$$
\begin{array}{rcccc}
 & 1_{(2,3)} & 1_{(2,1)} & 1_{(2,3} & 0_{(2,1)} \\
 & 1_{(2,3)} & 0_{(2,1)} & 0_{(3,3)} & 0_{(1,1)} \\
 & 0_{(3,3)} & 0_{(1,1)} & 1_{(3,3)} & 0_{(1,1)} \\
\cline{2-5}
B: & 0_{(2,2)} & 1_{(1,1)} & 0_{(3,2)} & 0_{(1,1)} \\
A: & 0_{(5,6)} & 0_{(1,2)} & 1_{(5,6)} & 1_{(1,2)} \\
T: & 1 & & &
\end{array}
$$

$$
\begin{array}{lll|lll|lll|lll}
\multicolumn{3}{l}{CELCS(B,0,0,12)} & \multicolumn{3}{l}{CELCS(A,1,1,8)} & \multicolumn{3}{l}{CELCS(B,2,2,4)} & \multicolumn{3}{l}{CELCS(A,3,16,0)} \\
 & 0_{(1,1)} & 1_{(1,1)} & & 0_{(2,2)} & 0_{(1,2)} & & 0_{(2,2)} & 1_{(1,1)} & & 0_{(5,6)} & 0_{(1,2)} \\
 & 1_{(1,1)} & 0_{(1,1)} & & 0_{(1,2)} & 0_{(2,2)} & & 1_{(1,1)} & 0_{(3,2)} & & 0_{(1,2)} & 1_{(5,6)} \\
 & 0_{(1,1)} & 1_{(1,1)} & & 0_{(2,2)} & 1_{(2,2)} & & 0_{(3,2)} & 0_{(1,1)} & & 1_{(5,6)} & 1_{(1,2)} \\
\cline{2-3}\cline{5-6}\cline{8-9}\cline{11-12}
B: & 1_{(1,1)} & 0_{(1,1)} & B: & 0_{(2,1)} & 1_{(2,1)} & B: & 1_{(1,1)} & 1_{(1,1)} & B: & 1_{(2,1)} & 0_{(1,1)} \\
A: & 0_{(2,1)} & 1_{(1,1)} & A: & 0_{(2,0)} & 0_{(1,0)} & A: & 0_{(3,2)} & 0_{(2,2)} & A: & 1_{(4,5)} & 0_{(3,5)} \\
T: & 1 & & T: & 2 & & T: & 1 & & T: & 2 & \\
\multicolumn{3}{l}{CELCS(B,0,0,14)} & \multicolumn{3}{l}{CELCS(B,1,1,10)} & \multicolumn{3}{l}{CELCS(B,2,2,6)} & \multicolumn{3}{l}{CELCS(B,3,4,2)} \\
S: & \multicolumn{2}{l}{1_{(1,1)}0_{(1,1)}} & S: & \multicolumn{2}{l}{0_{(2,1)}1_{(2,1)}} & S: & \multicolumn{2}{l}{1_{(1,1)}1_{(1,1)}} & S: & \multicolumn{2}{l}{1_{(2,1)}1_{(1,1)}} \\
CP: & \multicolumn{2}{l}{(0,16)} & CP: & \multicolumn{2}{l}{(1,12)} & CP: & \multicolumn{2}{l}{(2,8)} & CP: & \multicolumn{2}{l}{(3,4)} \\
 & & & & & & & & & \multicolumn{3}{l}{CELCS(A,5,16,0)} \\
 & & & & & & & & & S: & \multicolumn{2}{l}{1_{(4,5)}0_{(3,5)}} \\
 & & & & & & & & & CP: & \multicolumn{2}{l}{(5,2),(9,0)}
\end{array}
$$

Observe that in this example in the last step only at one position of the tree (the rightmost) both is possible, continuing with the string $B = 1_{(2,1)}0_{(1,1)}$ and continuing with the string $A = 1_{(4,5)}0_{(3,5)}$.

Remark 2 *In [10] the authors considered the more general concept of costed binary sequences, where any nonnegative real numbers are allowed as costs for changing a bit in the initial string of S. We emphasize that Algorithm 2 can also be applied to arbitrarily costed binary sequences.*

Similarly as pointed out in [10] for the case of sequences in $\mathcal{M}(x^{2^v} - 1)$ one can define a binary code $\mathcal{C}(v,u)$ to be the $(2^{v-u+1} - 2)$-dimensional subspace of $\mathcal{M}((x^2+x+1)^{2^v})$ containing all sequences of $\mathcal{M}((x^2+x+1)^{2^{v-u}-1})$ (or more generally $\bar{\mathcal{C}}(v,w) = \mathcal{M}((x^2+x+1)^w)$ for a positive integer $w \le v$).
The distance of a received sequence S (which we can identify with a binary vector of length 2^{v+1}) to the closest codeword is then the minimal value of k for which $L_k(S) \le 2^{v-u+1} - 2$. Evidently one obtains k from $CELCS(S)$.
For instance if we consider the code $\mathcal{C}(3,1)$ and receive the word 0110110011100011, then from Example 2 we see that the smallest k such that $L_k(S) \le 6$ is given by the critical point $(k, L_k(S)) = (3,4)$. Therefore at least 3 transmission errors occurred.
Without going into the details of decoding we remark that a decoding algorithm can be established similarly as for the code considered in [10]. For an efficient decoding algorithm for the code in [10] we refer the reader to [18, Algorithm 5.1].
Unique decoding depends of course on the minimum distance of the code $\mathcal{C}(v,u)$. The above considered code $\mathcal{C}(3,1)$ has minimum distance 4 and thus is a $[16,6,4]$ binary code, having a somewhat smaller information rate than the comparable $[16,7,4]$ binary code obtained in [10]. In general the calculation of the minimum distance for $\mathcal{C}(v,u)$ does not seem to be a straight forward adaption of the procedure in [12] giving the minimum distance of the $[2^n, 2^{n-u+1} - 1, 2^u]$ subcodes of the Reed-Miller code obtained in [10].

5 Conclusions

A lot of research has been performed on the linear complexity of d-periodic sequences over a finite field $\mathbb{F}_q$. Recently, in [8] a more general framework has been provided. Instead of the set of d-periodic sequences over $\mathbb{F}_q$, the set $\mathcal{M}(f)$ of sequences over $\mathbb{F}_q$ with a fixed characteristic polynomial $f \in \mathbb{F}_q[x]$ has been considered. In this context the set of d-periodic sequences is precisely the set $\mathcal{M}(f)$ with $f(x) = x^d - 1$.
In this contribution we initiate the development of algorithms for the determination of the linear complexity of a given sequence in $\mathcal{M}(f)$ and $f(x)$ is not of the form $x^d - 1$. We present algorithms for binary sequences with characteristic polynomial $f(x) = (x-1)^d$ and $f(x) = (x^2+x+1)^{2^v}$. Moreover a basic concept for establishing algorithms for binary sequences in $\mathcal{M}(f^{2^v})$, with $f(x) \in \mathbb{F}_2[x]$ irreducible is outlined.
The particular simplicity of the algorithm for $f(x) = (x^2 + x + 1)^{2^v}$ enables us to advance the algorithm to an algorithm for calculating the k-error linear complexity (spectrum). On the basis of the discussions in [10] we point to a possible application of this algorithm in coding theory.

The generalization of the Games-Chan algorithm to sequences over arbitrary finite fields presented in [5], suggests the possibility of a generalization of our considerations for binary sequences to sequences over arbitrary finite fields. In general, the new approach of considering the set of sequences $\mathcal{M}(f)$ with a given characteristic polynomial $f(x)$ may lead to a large variety of algorithms for analysing the linear complexity of sequences over a finite field $\mathbb{F}_q$.

Acknowledgement

The author would like to thank the anonymous referee for suggestions that helped to improve the paper.

References

[1] Chen, H.: Fast algorithms for determining the linear complexity of sequences over $GF(p^m)$ with period $2^t n$. IEEE Trans. Inform. Theory 51, 1854–1856 (2005)

[2] Chen, H.: Reducing the computation of linear complexities of periodic sequences over $GF(p^m)$. IEEE Trans. Inform. Theory 52, 5537–5539 (2006)

[3] Cusick, T. W., Ding, C., Renvall, A.: Stream Ciphers and Number Theory. North-Holland Publishing Co., Amsterdam, The Netherlands (1998)

[4] Dai, Z. D., Yang, J. H.: Linear complexity of periodically repeated random sequences. In: Advances in Cryptology - EUROCRYPT'91, (Davies, D. W., Ed.). Berlin: Springer-Verlag, 1991, Lecture Notes in Computer Science 547, 168–175 (1991)

[5] Ding, C.: A fast algorithm for the determination of linear complexity of sequences over $GF(p^m)$ with period p^n. In: The Stability Theory of Stream Ciphers. Berlin: Springer-Verlag 1991, Lecture Notes in Computer Science 561, 141–144 (1991)

[6] Ding C., Xiao, G., Shan, W.: The Stability Theory of Stream Ciphers. Lecture Notes in Computer Science 561. Springer-Verlag, Berlin, Germany (1991)

[7] Fu, F. W., Niederreiter, H., Su, M.: The expectation and variance of the joint linear complexity of random periodic multisequences. J. Complexity 21, 804–822 (2005)

[8] Fu, F. W., Niederreiter, H., Özbudak, F.: Joint linear complexity of multisequences consisting of linear recurring multisequences. Cryptography and Communications - Discrete Structures, Boolean Functions and Sequences, to appear

[9] Games, R. A., Chan, A. H.: A fast algorithm for determining the complexity of a binary sequence with period 2^n. IEEE Trans. Inform. Theory 29, 144–146 (1983)

[10] Lauder, G. B., Paterson, K. G.: Computing the linear complexity spectrum of a binary sequence of period 2^n. IEEE Trans. Inform. Theory 49, 273–280 (2003)

[11] Lidl, R., Niederreiter, H.: Finite Fields. Addison-Wesley, Reading, MA (1983)

[12] Massey, J. L., Costello, D. J., Justesen, J.: Polynomial weights and code constructions. IEEE Trans. Inform. Theory 19, 101–110 (1973)

[13] Meidl, W.: Reducing the calculation of the linear complexity of $u2^v$-periodic binary sequences to Games-Chan algorithm. Designs, Codes, Cryptogr. 46, 57–65 (2008)

[14] Meidl, W., Niederreiter, H.: Linear complexity, k-error linear complexity, and the discrete Fourier transform. J. Complexity 18, 87–103 (2002)

[15] Meidl, W., Niederreiter, H.: On the expected value of the linear complexity and the k-error linear complexity of periodic sequences. IEEE Trans. Inform. Theory 48, 2817–2825 (2002)

[16] Meidl, W., Niederreiter, H.: The expected value of the joint linear complexity of periodic multisequences. J. Complexity 19, 61–72 (2003)

[17] Niederreiter, H.: Sequences with almost perfect linear complexity profile. In: Advances in Cryptology - Proceedings of EUROCRYPT 1987, (Chaum, D., Price, W. L., Eds.). Berlin: Springer-Verlag, 1988, Lecture Notes in Computer Science 304, 37–51 (1988)

[18] Salagean, A.: On the computation of the linear complexity and the k-error linear complexity of binary sequences with period a power of two. IEEE Trans. Inform. Theory 51, 1145–1150 (2005)

[19] Stamp, M., Martin, C. F.: An algorithm for the k-error linear complexity of binary sequences with period 2^n. IEEE Trans. Inform. Theory 39, 1398–1401 (1993)

[20] Xiao, G., Wei, S., Lam, K. Y., Imamura, K.: A fast algorithm for determining the linear complexity of a sequence with period p^n over $GF(q)$. IEEE Trans. Inform. Theory 46, 2203–2206 (2000)

[21] Xiao, G., Wei, S.: Fast algorithms for determining the linear complexity of period sequences. In: Progress in Cryptology - INDOCRYPT 2002 (Menezes, A., Sarkar, P., Eds.). Berlin: Springer-Verlag, 2002, Lecture Notes in Computer Science 2551, 12–21 (2002)