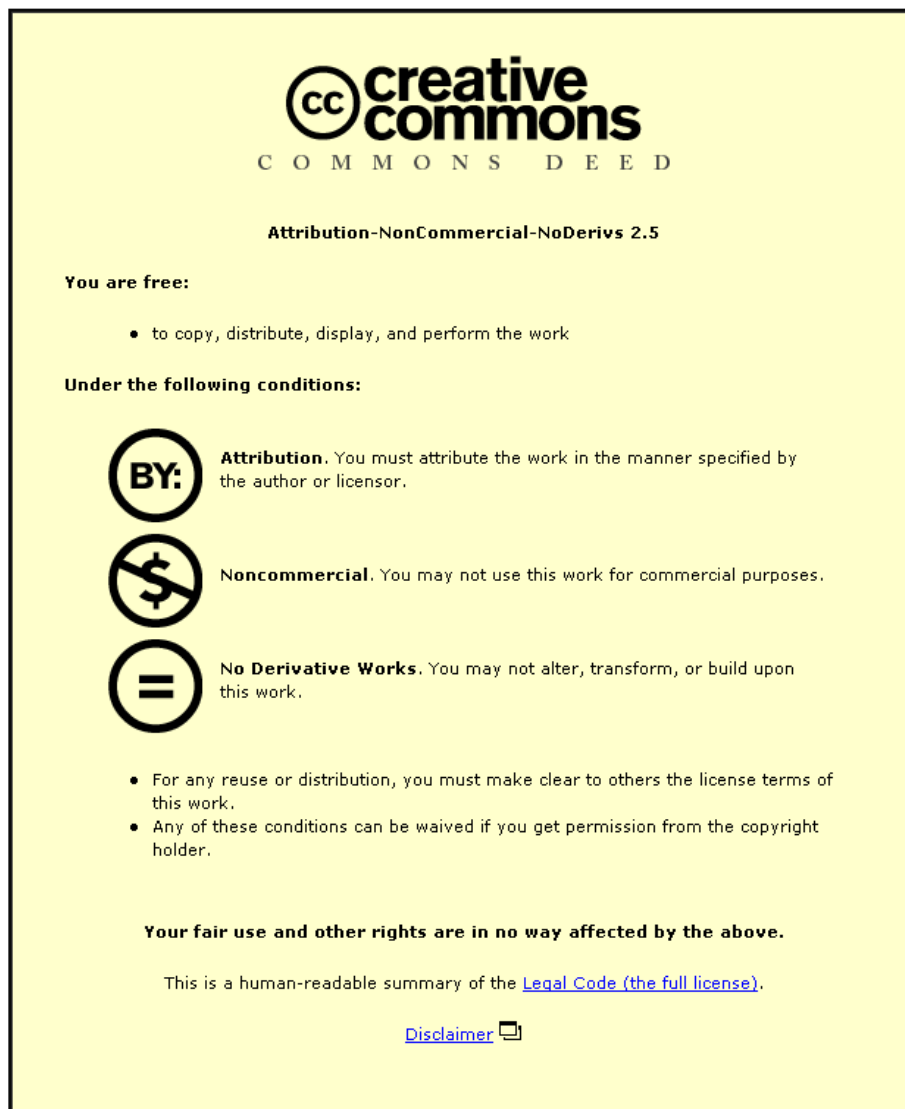




This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (<https://dspace.lboro.ac.uk/>) under the following Creative Commons Licence conditions.



CC creative commons
COMMONS DEED

Attribution-NonCommercial-NoDerivs 2.5

You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

BY: **Attribution.** You must attribute the work in the manner specified by the author or licensor.


Noncommercial. You may not use this work for commercial purposes.

No Derivative Works. You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the [Legal Code \(the full license\)](#).

[Disclaimer](#) 

For the full text of this licence, please go to:
<http://creativecommons.org/licenses/by-nc-nd/2.5/>

Complexity Measures for Classes of Sequences
and Cryptographic Applications

by

Alex Burrage

A Doctoral Thesis

Submitted in partial fulfilment
of the requirements for the award of

Doctor of Philosophy
of
Loughborough University

1st September 2012

Copyright 2012 Alex Burrage

Abstract

Pseudo-random sequences are a crucial component of cryptography, particularly in stream cipher design. In this thesis we will investigate several measures of randomness for certain classes of finitely generated sequences.

We will present a heuristic algorithm for calculating the k -error linear complexity of a general sequence, of either finite or infinite length, and results on the closeness of the approximation generated.

We will present an linear time algorithm for determining the linear complexity of a sequence whose characteristic polynomial is a power of an irreducible element, again presenting variations for both finite and infinite sequences. This algorithm allows the linear complexity of such sequences to be determined faster than was previously possible.

Finally we investigate the stability of m-sequences, in terms of both k -error linear complexity and k -error period. We show that such sequences are inherently stable, but show that some are more stable than others.

Acknowledgements

The research in this thesis was undertaken as a joint work with Ana Sălăgean and Raphael Phan, and I am indebted to them for all their help and support. I would also like to thank my fellow PhD students Dave Gardner and Richard Winter for their thoughts and opinions on my work and Honggang Hu for providing access to a copy of [37]. Finally, I would like to thank my parents, and especially my sister Clare, for their love and support.

Although all of the core results that are presented here were originally developed by myself, they were developed and refined through an iterative process with my main supervisor (Ana Sălăgean), and so the credit for the final form of these results must be partly split between us.

*What we call chaos is just patterns we havent recognized. What we call random
is just patterns we can't decipher.*
– Chuck Palahniuk

Contents

Abstract	ii
Acknowledgements	iii
1 Introduction	1
1.1 One Time Pad	2
1.2 Pseudo-Random Sequences	3
1.3 Pseudo-Random Sequence Properties	5
1.4 Modern Stream Cipher Proposals	6
2 Definitions and Preliminary Results	8
3 Literature Review	16
3.1 Linear Complexity	16
3.2 k -Error Linear Complexity	17
3.3 Approximation Algorithms	19
3.4 General Results on k -Error Linear Complexity	19
3.5 General k -Error Knowledge	20
3.6 Determining the Required Number of Errors	21
3.7 Feedback Carry Shift Registers	22
3.8 Irreducible Cyclic Codes	22
3.9 Stream Cipher Proposals	23
3.10 Contribution to Knowledge	24
4 A Heuristic Algorithm for Determining k-error Linear Complexity of a Sequence	27
4.1 Introduction	27
4.2 The Algorithm	27
4.3 Notes on the Algorithm	28
4.4 Performance of the Algorithm	30
4.5 Numerical Results	35
4.6 Alternative Implementation	36

4.7	Numerical Results	40
4.8	Analysis of Results	42
5	Linear Complexity for Sequences with Characteristic Polynomial of the Form f^v for an Irreducible f	43
5.1	Linear Complexity of Infinite Sequences with a Characteristic Polynomial a Power of an Irreducible Polynomial	44
5.2	Linear Complexity of Finite Sequences with Characteristic Polynomial a Power of an Irreducible Polynomial	50
5.3	Linear Complexity of Infinite Sequences with a Characteristic Polynomial a Product of Known Irreducible Factors	54
6	m-Sequence Stability	56
6.1	k -Error Linear Complexity and Period for Various Classes of m-Sequence	57
6.1.1	Prime period	58
6.1.2	Reducing the period of an m-sequence by an arbitrary factor	59
6.1.3	Reducing the period by a Mersenne number	62
6.1.4	Reducing the period by a prime p with $\text{ord}_p(2) = p - 1$	63
6.1.5	The minimum number of errors needed for reducing the period of an m-sequence	65
6.2	Application to Grain and other Stream Ciphers	68
7	Further results on m-sequence stability	72
7.1	Period Stability Results	72
7.1.1	Two Weight Irreducible Cyclic Codes	72
7.1.2	Prime factors N , such that $N \equiv 3 \pmod{4}$ and $\text{ord}_N(2) = (N - 1)/2$	74
8	Conclusions	77
8.1	Future Work	78
	References	80
A	k-Error Period Tables	87

List of Tables

4.1	Example Running of <i>kErAp</i>	35
4.2	Size of the search Space, N=32	36
4.3	Size of the search Space, N=48	37
4.4	Size of the Search Space, N=60	37
4.5	Numerically Generated Results, on 50 sequences, each of length 30 .	39
4.6	Size of the search Space, N=32	40
4.7	Size of the search Space, N=48	41
4.8	Size of the Search Space, N=60	41
4.9	Numerically Generated Results, on 50 sequences, each of length 30 .	42
5.1	Example Running of LinComp	50
5.2	Example running of LinCompFinite	52
6.1	$\text{ord}_p(2)$ for small prime p	67
6.2	Proportion of Mersenne numbers with certain factors	67
6.3	Errors Required for Reducing the Period of the Grain LFSR	69
6.4	Errors Required for Reducing the Period of the DECIM ^{v2} LFSR . . .	71
A.1	m-Sequence Period Stability Results	90

List of Corrections

Chapter 1

Introduction

Cryptography as a discipline is concerned with the problem of secure communications in the presence of untrusted third parties. Broadly speaking, it searches for ways to transmit data such that the set of participants that can obtain the data contains exactly those parties who we wish it to and no more. Conversely, the related discipline of cryptanalysis is the study of how to extract information from communications which are designed to make such an abstraction difficult or impossible. To put this in a more concrete setting we consider a game, or thought experiment, consisting of three parties, Alice, Bob and Eve. Alice has some information that she wishes to send to Bob. However, the channel between Alice and Bob is unsecure - that is, it is possible for a third party, Eve, to intercept the message. Therefore Alice has to encrypt the message in such a way that Bob is able to decrypt it (and hence recover the original message) but Eve cannot. This is done through the use of a shared secret between Alice and Bob, which Eve does not possess, usually in the form of a key. We refer to the message that Alice wishes to translate as the plaintext, and the encrypted information (which we assume Eve can obtain) as the ciphertext. We say that an encryption method is secure if Alice can trust that given the ciphertext, only Bob can obtain the plaintext.

There are various different attributes we can give to each of the parties in the game, and to the channel itself, and each combination gives rise to a different model of security. One of the most important distinctions is in the way that the plaintext becomes available, and we distinguish between stream and block ciphers. If we assume that Alice possesses the entire message when she begins her encryption, it is possible for her to employ methods which act on the entirety of the plaintext, or at least break it up into large blocks, and then perform encryption on each of these. Bob will then receive large chunks of ciphertext, and decrypt each block as a whole. Schemes that operate in this way are known as block ciphers. Conversely, consider the case where Alice needs to begin to communicate the message to Bob whilst the plaintext is still being generated (or becoming

available). In this scenario, she must encrypt the data on-the-fly, in very small pieces, as any waiting to obtain more of the plaintext is either not possible or not acceptable. Bob will then receive the ciphertext piece-by-piece and begin to decrypt each piece as he is still receiving more of the ciphertext. Schemes that operate in this scenario are known as stream ciphers. For some situations, it is possible to apply a block cipher in a mode whereby it approximates the behavior of a stream cipher, but such systems always suffer from an increase in latency.

Arguably the second most important distinction between types of encryption schemes is between symmetric and asymmetric cryptography. As mentioned before, the only situation whereby Bob can convert the ciphertext back into plaintext but Eve cannot, is if Bob possesses some secret piece of information that Eve does not (known as a key), and it is this that Alice exploits. Classically, it was assumed that Alice possesses the same key as Bob, and this scenario is known as symmetric encryption as the knowledge at either end of the transmission is identical. However, in the last 40 years, a new scenario has been developed, whereby Alice and Bob do not possess the same information. Rather, Alice only possesses Bob's public key, which she can use to encrypt the plaintext, and then Bob can use a different key - his private key - to decrypt the data. The crucial fact here is that Alice does not possess Bob's private key, in fact, no one other than Bob does. This is very useful for situations whereby there are many participants who wish to send secret messages: Bob is free to make his public key widely known to anyone who wishes to send a message to him. Then, he does not have to worry about Eve obtaining this information as it is not possible to decrypt a ciphertext by using only the public key.

Whilst there has been a great deal of progress made in the study of block ciphers, there is still a comparative lack of maturity in the field of stream ciphers, from both cryptographic and cryptanalytic viewpoints. We have looked at the theoretical aspects of symmetric stream ciphers, and seen how some of these can be applied to practical schemes.

1.1 One Time Pad

Virtually all stream ciphers that are in use today, or have been in use at some point can trace their design back to the one time pad, or Vernam cipher, first presented in 1882 by Miller [57]. It uses a very simple encryption method, and gains its security from the properties of its key, as follows. Assume that Alice and Bob have agreed on some alphabet and a structure over it that allows addition of its elements. Now, assume Alice has a message m consisting of n symbols from

her chosen alphabet:

$$m = m_0, m_1, \dots, m_{n-1}$$

Further, assume that both she and Bob possess a secret key, k , which is also n symbols from the chosen alphabet:

$$k = k_0, k_1, \dots, k_{n-1}$$

Then to encrypt her message, Alice simply adds each symbol of the message to the corresponding symbol of the key (using addition as defined over her alphabet) to obtain the n symbols of the ciphertext, c :

$$\begin{aligned} m \oplus k &= m_0 \oplus k_0, m_1 \oplus k_1, \dots, m_{n-1} \oplus k_{n-1} \\ &= c_0, c_1, \dots, c_{n-1} \\ &= c \end{aligned}$$

Then, when Bob receives the ciphertext c , he can recover the plaintext p by subtracting each term from the key from the corresponding ciphertext term. Although this scheme may seem simple, it was proven by Shannon in [68] that if the terms of the key are random, and if each key defines a unique bijection when combined with the operator \oplus , then the system is perfectly secure. By this, it is meant that if Eve intercepts a ciphertext c (and she does not have any information about the key), no information is leaked about what the plaintext was - that is, the actual plaintext that was encrypted is as likely to have been used as any other (of suitable length). Whilst this property gives this scheme great strength, the scheme is very seldom used for the very same reason: it is necessary to possess a key that is truly random and as long as the message. For any reasonable message length, ensuring that Alice and Bob have a long enough key (that they have stored in some secure way) becomes very difficult. Therefore, virtually all research into stream ciphers has been into ways that aim to get as close as possible to replicating the security provided by the one time pad, whilst using shorter keys, and hence increasing the practicality of the system.

1.2 Pseudo-Random Sequences

The theoretical strength of the one-time pad, combined with its impracticality, has led to the following structure being adopted for virtually all stream cipher

proposals. Alice and Bob each possess a small key, k . By feeding this key into some deterministic function f , they generate identical pseudorandom sequences $f(k) = k_0, k_1, k_2, \dots$. Then encryption on the plaintext is done term-by-term just as for the one-time pad, and since Bob possesses the same sequence, he can decrypt, just as for the one-time pad as well. Assuming that the set of keys is large enough, it is easy to see that the strength of this scheme depends on the function used to generate the secret sequence, namely, how close to random the sequence is. If it is very easy to predict the terms of the sequence, then the scheme will not capture the security of the one-time pad, and will be easy to attack. Conversely, if the sequence appears to be random, and its terms cannot be predicted, then the system will be secure, and no attacks will be possible. Therefore, research into stream ciphers is closely bound with research into the randomness properties of sequences. The important fact in this is that any finitely generated sequence (as Alice and Bob only possess finite keys, and use finite mechanisms to generate the sequence) must be ultimately periodic, that is, after some initial sequence (possibly of zero length) the sequence must begin repeating. This is because there are only finitely many states the system can enter, and since the process is deterministic (which it needs to be for Alice and Bob to generate the same sequences) if the system enters a state it has already been in, all future behavior will be identical. Therefore, whatever generating function is used, the output sequence must be ultimately periodic, and hence predictable, and hence insecure. It is the cryptographer's role to try to find the functions which minimise the degree of predictability, and the cryptanalyst's role to try to maximise it.

One basic way to generate a long, pseudo-random sequence given a small item of starting data is by using a linear feedback shift register (LFSR). To simplify the following discussion, we will assume that the sequence we are generating is over the finite field K . An LFSR consists of a series of n registers, each of which is capable of holding a single value from K , and a characteristic function, which acts on K^n and outputs a value in K . Initially we fill the n registers with the n values which make up the secret key. Then, at each clock step, the registers are updated using the following function:

$$\begin{aligned} s_i(t+1) &= s_{i+1}(t) \quad \forall i \leq n-2 \\ s_{n-1}(t+1) &= f(s_0(t), s_1(t), \dots, s_{n-1}(t)) \\ &= \sum_{i=0}^{n-1} s_i(t) a_i \end{aligned}$$

where $s_i(t)$ is the contents of register i at time t , f is the characteristic function

and the a_i are constants. Then the sequence $(s_0(0), s_0(1), s_0(2), \dots)$ is the sequence that can be used for encrypting the message.

LFSRs are by far the most widely studied way of generating a pseudo-random sequence. The reason for this is that any sequence that is finitely generated (that is, any ultimately periodic sequence) can be generated by some LFSR. Further, we have a very efficient way of finding the simplest LFSR that can generate a sequence. Therefore, analysis of sequences is often studied in terms of the LFSR that can generate the sequence, regardless of the way the sequence was actually generated. This allows different sequences to be compared, and it is therefore easy to determine which possess the best properties for the intended use.

1.3 Pseudo-Random Sequence Properties

To determine the strength of a pseudo-random sequence, and hence its suitability for use, we study its properties. The simplest of these is the minimal period length of the sequence. That is, the shortest number of terms over which the sequence repeats. Since stream cipher encryption is based on such a simple encryption method (a term by term addition) all of the strength of the encryption is derived from the secrecy of the sequence. However, if the pseudo-random sequence repeats, an attacker knows that any terms separated by the minimal period length were encrypted by addition of the same term. Since this is true for all terms after the sequence has begun to repeat, a huge amount of information is leaked, and virtually all security is lost. Therefore it is a very basic requirement of stream ciphers that the minimal period of the pseudo-random sequence that is used for encryption is longer than the message length. Unless the minimal period length of a sequence is very long, therefore, the sequence will not be suitable for use in encryption, and so it is necessary to determine the minimal period length of a sequence (or at the very least a lower bound on its value) before a sequence can be used in a cipher scheme.

Another basic property of a pseudo-random sequence is its linear complexity. This is defined as the shortest LFSR that is capable of generating the sequence. If a sequence has a small linear complexity, say n , then it can be completely determined by a small amount of knowledge, namely, the n coefficients of the characteristic polynomial, and any consecutive n terms of the sequence. For a sequence of period m , the linear complexity can be as small as $\log(m)$, and so the entire minimal period (and hence the entire sequence) can be generated from a very small amount of knowledge. Therefore it is a basic requirement of a pseudo-random sequence to be used in a stream cipher that it have a large linear complexity.

From these two basic properties, we can derive two more, the k -error period

and the k -error linear complexity. These are the minimal period and the minimal linear complexity of any sequence that can be generated by altering up to k terms in a period of the sequence. To give a simple example, the following sequence has a period length of 6: 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, ... However, by changing just one term in each period - the final one which we change from a 0 to a 1 - the sequence becomes: 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, ... which has a minimal period of just 3. Therefore while the minimal period of this sequence was 6, the 1-error period was just 3. The reason that this is so important for cryptographers, is that if a sequence has a small k -error period or k -error linear complexity, and if an attacker can efficiently find the sequence with shorter period or reduced linear complexity, they may approximate the original sequence by a simpler one, if they are prepared to tolerate up to k errors in their decryption. This may transform the sequence from one that was previously considered secure into one that is no longer so, and so it is vital for security purposes that the k -error period and linear complexity are known, or at the very least a lower bound on their values is established.

It should be noted that there are other, more complex properties of pseudo-random sequences such as balance and mixing, but they will not be discussed in this thesis

1.4 Modern Stream Cipher Proposals

Modern stream ciphers are highly complex and often take many years of analysis to fully understand. They are typically made up of several smaller blocks (known as primitives) whose outputs are combined in complex ways to form the output of the sequence as a whole. They are designed so that this combining process retains the desirable properties from each primitive, but removes any weaknesses that each particular primitive might have. To give a simplified example, consider a case where we have just two primitives, A and B . Assume, for the sake of argument, that A has a high k -error period, but a low k -error linear complexity, and that B has a high k -error linear complexity, but a low k -error period. Then, if the outputs of the two primitives could be combined in such a way that the k -error period was the sum of the two primitives and likewise the k -error linear complexity was the sum of the two primitives, we would have a sequence with high k -error period, and high k -error linear complexity. In reality the k -error period and k -error linear complexity are so closely related that this situation is unlikely to occur, but it is useful as an illustrative example. To further increase the security, the outputs of each primitive are often fed back into the generating system, and used as inputs to the other primitives. In this way, the designers can

further increase the interdependence of the proposal, and ensure that it is much harder to isolate one primitive for study, and any potential weaknesses present in a primitive are not present in the output of the proposal.

Currently there is a considerable amount of effort going in to generating new proposals, and studying them to determine their security. The recent NESSIE and ESTREAM projects were two collaborative efforts by the community to find stream ciphers with the desirable properties, and their work is still ongoing. There are also many individual proposals in the literature. The topics discussed in this thesis are applicable only to the primitives used in stream ciphers, as is the usual way of studying such designs, and it is hoped that the results presented are highly useful to stream cipher designers, as well as to those studying existing designs.

Chapter 2

Definitions and Preliminary Results

Notation 2.1. Let K be a finite field. We denote by $K[x]$ the set of all polynomials over K .

Definition 2.2. Let K be a finite field. Given a degree n monic polynomial $f = x^n + c_{n-1}x^{n-1} + \dots + c_0 \in K[x]$ and n initial values, $s_0, s_1, \dots, s_{n-1} \in K$, an infinite sequence $s = (s_0, s_1, \dots)$ is recursively generated by using the following linear recurrence relation:

$$s_i = -c_0s_{i-n} - c_1s_{i-n+1} - \dots - c_{n-1}s_{i-1}$$

for $i \geq n$. Note that it is possible to generate identical sequences using different polynomials. We refer to f as a characteristic polynomial of s , and any sequence which can be generated in this way as a linear recurrent sequence. Equivalently we can use f to generate a finite sequence $s' = (s_0, s_1, \dots, s_{m-1})$ by only considering the first m terms of s . Note that elsewhere in the literature the term feedback polynomial is sometimes used instead of characteristic polynomial. The two are entirely interchangeable.

Definition 2.3. For any linear recurrent sequence s , a characteristic polynomial of the lowest degree is referred to as the minimal polynomial, and its degree the linear complexity of the sequence. We will denote the linear complexity of s as $\text{LC}(s)$.

Notation 2.4. We will denote the all zero sequence as $\mathbf{0}$. We will use the same notation for the infinite and all finite sequences, where the length of the sequence will be clear from the context.

Note that by convention $\mathbf{0}$ has linear complexity 0, regardless of its length.

Lemma 2.5. *For infinite sequences the minimal polynomial is unique and any other characteristic polynomial is a multiple of the minimal polynomial.*

Notation 2.6. *For a polynomial f we denote by $\text{wt}(f)$ the weight of f , i.e. the number of non-zero coefficients of f (by analogy with the Hamming weight of vectors).*

For convenience we will introduce the following notation:

Definition 2.7. *Let K be a finite field. Let f be a monic polynomial in $K[x]$. We define $\mathcal{M}(f)$ to be the set of all infinite sequences over K with a characteristic polynomial equal to f . We also define $\mathcal{M}(f^\infty) = \cup_{i=0}^{\infty} \mathcal{M}(f^i)$.*

The following definition is a commonly used notion:

Definition 2.8. *Let K be a finite field. Let $f = \sum_{i=0}^n a_i x^i$ be a polynomial in $K[x]$, and $s = s_0, s_1, \dots$ an infinite sequence over K .*

We define the action of f on s , denoted fs , to be the infinite sequence $t = t_0, t_1, \dots$ defined by $t_i = \sum_{j=0}^n a_j s_{i+j}$.

For a finite sequence $s' = (s_0, s_1, \dots, s_{m-1})$ over K with $m > n$ we define the action of f on s' , denoted fs' , to be the finite sequence $t' = (t_0, t_1, \dots, t_{m-n-1})$ defined by $t_i = \sum_{j=0}^n a_j s_{i+j}$. (One could extend the definition to $m \leq n$ but this situation will not occur in this thesis.)

Definition 2.9. *A sequence s is called periodic if there exists a positive integer t such that $s_i = s_{i+t}$ for all $i = 0, 1, \dots$. We call t a period of s and we call the smallest positive such t the minimal period of s and denote it by $P(s)$.*

Using the terminology of actions, the following results concerning characteristic polynomials are immediate:

Lemma 2.10. *Let K be a finite field. Let $f \in K[x]$ be monic and let s be an infinite sequence over K . Then:*

(i) f is a characteristic polynomial of s iff $fs = \mathbf{0}$. Moreover, f is the minimal polynomial of s iff f is a polynomial of minimal degree for which $fs = \mathbf{0}$.

(ii) Let $g \in K[x]$. Let $\text{gcd}(f, g) = f_2$ and $f = f_1 f_2$ with all polynomials monic. If f is the minimal polynomial of s then f_1 is the minimal polynomial of gs .

(iii) If s is periodic and N is a period of s then N is also a period of fs .

Proof. Parts (i) and (iii) are clear. For (ii) write g as $f_2 g'$. Denote the minimal polynomial of gs by f_3 . We will prove that $f_1 = f_3$. From (i) we know $fs = f_1 f_2 s = \mathbf{0}$ and $f_3 gs = f_3 f_2 g' s = \mathbf{0}$. Since the minimal polynomial of a sequence divides any other characteristic polynomial, $f_1 f_2 | f_3 f_2 g'$. Since g' and f_1 are coprime we have

$f_1|f_3$. On the other hand, $fs = \mathbf{0}$ implies $fg's = f_1f_2g's = f_1gs = \mathbf{0}$ and since f_3 is the minimal polynomial of gs , we have $f_3|f_1$, which combined with $f_1|f_3$ proved previously results in $f_3 = f_1$. \square

Based on the well-known exponentiation rule (sometimes known as ‘‘Freshman’s dream’’): $(a + b)^p = a^p + b^p$ for all $a, b \in K$ and p is the characteristic of K , we have:

Lemma 2.11. *Let K be a finite field of characteristic p , and let $f \in K[x]$. Then $\text{wt}(f^{p^w}) = \text{wt}(f)$ for any $w \geq 0$.*

Definition 2.12. *The degree of a polynomial $f = \sum_{i=0}^n a_i x^i$ is the power of its largest non-zero term, n , and we will denote this as $\text{deg}(f)$.*

Definition 2.13. *The order of a polynomial f (with $x \nmid f$), denoted $\text{ord}(f)$, is the smallest integer n such that f is a factor of $x^n - 1$.*

Lemma 2.14. *([44, Theorem 6.27]) The minimal period of a periodic sequence is the same as the order of its minimal polynomial.*

Lemma 2.15. *Let K be a finite field of characteristic p . The order of an irreducible polynomial $f \in K[x]$ is a factor of $p^{\text{deg}(f)} - 1$, and hence $\text{ord}(f)$ is not divisible by p .*

The order of a power of an irreducible polynomial can be derived as follows:

Theorem 2.16. *([44, Theorem 3.8]) Let f be irreducible over $K[x]$ and let r be a positive integer. Then $\text{ord}(f^r) = p^t \text{ord}(f)$ where t is the smallest integer with $p^t \geq r$, and p is the characteristic of the underlying field.*

Notation 2.17. *Consider a finite sequence, s . We denote by $\text{wt}(s)$ the Hamming weight of s , that is, the number of non-zero terms in s . For simplicity, we will usually just refer to the weight of s .*

Definition 2.18. *Consider an infinite, periodic sequence, s . Once we have specified a (not necessarily minimal) period of s , m , we can refer to the weight of s by the weight of the finite sequence s_0, s_1, \dots, s_{m-1} .*

Note that the weight of a sequence is not unique as it is dependent on the length of the period being considered.

Definition 2.19. *Consider two finite sequences, s and s' , of the same length, m . Then we define the Hamming distance, $d(s, s')$, between the two sequences as the weight of the sequence $s_0 - s'_0, s_1 - s'_1, \dots, s_{m-1} - s'_{m-1}$. For simplicity, we will usually just refer to the distance between the two sequences.*

Definition 2.20. Consider two infinite sequences which share a period, m . Then we define the distance between the two sequences as the distance between the two finite sequences which consist of the first m terms of each.

Note that the two sequences do not need to share a minimal period, and that the distance between them is not uniquely defined, as it is dependent on the period.

Notation 2.21. We will denote the Galois Field of order n as $\text{GF}(n)$.

Definition 2.22. Let $\alpha \in K$ be a generator of the multiplicative group of K , that is for every element β in the multiplicative group of K , there exists a positive integer i such that $\alpha^i = \beta$. Then α is a primitive element and its minimal polynomial is a primitive polynomial.

Definition 2.23. Consider a linear recurrent sequence over $\text{GF}(2)$ with a primitive characteristic polynomial from $\text{GF}(2)[x]$ and initial values that are not all zero. Any such sequence is known as an m -sequence and has period equal to $2^n - 1$ where n is the degree of the characteristic polynomial.

Lemma 2.24. For any binary sequence generated by a polynomial of degree n , the period is no greater than $2^n - 1$ and the maximum is achieved exactly when the sequence is an m -sequence.

The k -error linear complexity of a sequence is a parameter that generalizes the linear complexity:

Definition 2.25. [69] [22] Let s be an infinite periodic sequence with period m and $0 \leq k \leq m$. The k -error linear complexity of s is defined as:

$$\text{LC}_k(s) = \min\{\text{LC}(s') : s' \text{ sequence of period } m, d(s, s') \leq k\}.$$

Note that this definition implicitly depends on the period.

Definition 2.26. [43] The error linear complexity spectrum of a sequence s of period m is a list of pairs, $(k, \text{LC}_k(s))$, where k takes all values in the range $0 \leq k \leq \text{wt}(s)$. A critical point in the spectrum is where $\text{LC}_k(s) < \text{LC}_{k-1}(s)$ for $k > 1$ or $k = 0$.

Note that knowing the critical points of the error linear complexity spectrum is enough to generate the whole spectrum.

Lemma 2.27. Let s be an infinite periodic sequence of period m . Then $\text{LC}_{\text{wt}(s)}(s) = 0$, so the last critical point in the complexity spectrum is $(\text{wt}(s), 0)$. If s is a binary sequence, $\text{LC}_{m-\text{wt}(s)}(s) \leq 1$ so if $\text{wt}(s) > m/2$ the penultimate critical point in the spectrum is $(m - \text{wt}(s), 1)$.

We will need a few other parameters related to k -error linear complexity:

Definition 2.28. *The minimum number of errors required in each minimal period to reduce the linear complexity of s to a value that is at most c will be denoted by $E_{LC(s)}(c)$. The smallest number of errors needed to strictly reduce the linear complexity will be called the linear complexity stability threshold and will be denoted $E_{LC(s)}$.*

Note that for any fixed sequence s , if we consider $E_{LC(s)}(c)$ as a function of c , and $LC_k(s)$ as a function of k , then $E_{LC(s)}(c)$ is the minimum of the preimage of c under $LC_k(s)$.

Example 2.29. *Consider the binary sequence s whose minimal period is*

$$(0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0)$$

Then s has the following linear complexity spectrum: $(0, 15)$, $(1, 15)$, $(2, 10)$, $(3, 10)$, $(4, 5)$, $(5, 5)$, $(6, 2)$, $(7, 2)$, $(8, 0)$. The critical points on this spectrum are $(0, 15)$, $(2, 10)$, $(4, 5)$, $(6, 2)$, $(8, 0)$ and the linear complexity stability threshold is 2. We also have, for example, $E_{LC(s)}(5) = 4$.

We define for the period of a sequence analogues of the k -error linear complexity parameters:

Definition 2.30. *For an infinite periodic sequence s we define the k -error period to be:*

$$P_k(s) = \min\{P(t) : t \text{ sequence of period } P(s), d(s, t) \leq k\}$$

Note t above must have (possibly not minimal) period equal to $P(s)$.

The minimum number of errors required in each minimal period to reduce the minimal period of s to at most c will be denoted by $E_{P(s)}(c)$. The smallest number of errors needed to strictly reduce the period will be called the period reduction value and will be denoted $E_{P(s)}$ (i.e. $E_{P(s)} = E_{P(s)}(P(s) - 1)$).

Note that $E_{P(s)}(c)$ and $P_k(s)$ have the same relation as $E_{LC(s)}(c)$ and $LC_k(s)$.

We recall some terminology from number theory.

Definition 2.31. *A Mersenne number is any number of the form $2^n - 1$ for some positive integer n . A Mersenne prime is a Mersenne number that is prime.*

Note that in the definition of a Mersenne number we do not require that either n or $2^n - 1$ be prime, by following the terminology used in, for example, the Online Encyclopedia of Integer Sequences [60].

Definition 2.32. For a, q relatively prime positive integers we have $a^{\varphi(q)} \equiv 1 \pmod{q}$, where φ denotes Euler's totient function. The multiplicative order of a modulo q is the smallest integer u such that $a^u \equiv 1 \pmod{q}$. We will refer to this as simply the order of $a \pmod{q}$ and denote it as $\text{ord}_q(a)$.

Lemma 2.33. For any integer v , $a^v \equiv 1 \pmod{q}$ iff $\text{ord}_q(a) | v$. In particular $\text{ord}_q(a) | \varphi(q)$. We have $\log_a(q+1) \leq \text{ord}_q(a) \leq \varphi(q) \leq q-1$. When $q = a^n - 1$ the lower bound is reached. The upper bound $\text{ord}_q(a) = q-1$ is also reached for certain values of a and q . Namely, we have $\varphi(q) = q-1$ iff q is prime. For each prime q , there is at least one value a which achieves $\text{ord}_q(a) = q-1$ (namely the primitive elements in the field of integers modulo q).

Definition 2.34. A code, \mathcal{C} , of length n over K is a set of words $c = (c_0, c_1, \dots, c_{n-1})$. We will associate with this codeword the polynomial $c(x) = c_0 + c_1x + \dots + c_{n-1}x^{n-1}$ for ease of notation.

Definition 2.35. A code is linear if the codewords define a linear subspace of K^n .

Definition 2.36. A code \mathcal{C} is cyclic if it is linear and if any cyclic shift of a codeword is also a codeword, i.e., whenever $(c_0, c_1, \dots, c_{n-1})$ is in \mathcal{C} then so is $(c_{n-1}, c_0, \dots, c_{n-2})$.

Definition 2.37. Let R_n denote the ring $K[x]/(x^n - 1)$. An ideal \mathcal{C} of R_n is a linear subspace of R_n such that if $c(x) \in \mathcal{C}$ then so is $xc(x)$. For some polynomial f which we will call the generating polynomial of the ideal, the ideal is equal to all polynomials fg for any $g \in R_n \neq 0$.

Note that if the elements of R_n are identified with vectors over K of length n in the obvious way then R_n being an ideal is equivalent to the corresponding code being cyclic.

Definition 2.38. A minimal ideal is one which does not contain any smaller nonzero ideal. The corresponding cyclic code is called a minimal cyclic code and we refer to the generating polynomial of the ideal as a generating polynomial of the code.

An alternative definition for a set of linear codes is given below (presented only in the binary case). We will go on to show that the set of codes satisfying this definition is exactly the same as the set of minimal cyclic codes.

Definition 2.39. The absolute trace function of $\beta \in \text{GF}(2^m)$ is defined as $\sum_{i=0}^{m-1} \beta^{2^i}$ and is denoted $\text{Tr}(\beta)$. As this is the only trace function we will use in this thesis, we will simply refer to it as the trace function.

Definition 2.40. Let $r = 2^m$ for some integer m , and let N be an integer dividing $r - 1$. Put $n = (r - 1)/N$, let α be a primitive element of $\text{GF}(r)$ and $\theta = \alpha^N$. Then the set

$$\mathcal{C}(r, N) = \{\text{Tr}(\beta), \text{Tr}(\beta\theta), \dots, \text{Tr}(\beta\theta^{n-1}) : \beta \in \text{GF}(r)\}$$

is an irreducible cyclic $[n, m]$ binary code.

In the following lemma we will use the term *reciprocal* to mean reversing the order of the coefficients of a polynomial. That is if $f = c_0 + c_1x + \dots + c_nx^n$ is a polynomial, the reciprocal of f is: $c_n + c_{n-1}x + \dots + c_0x^n$.

Lemma 2.41. Let s be an m -sequence generated by characteristic polynomial f . If we consider the decimated sub-sequences of s as codewords from a code \mathcal{C} then \mathcal{C} is an irreducible cyclic code with check polynomial equal to the reciprocal of f (where a check polynomial is a polynomial g such that $gc = 0$).

Proof. The terms of s can be written as $s_i = \sum_j a^{2^j} (\theta^{2^j})^i$ where a is uniquely determined by the initial values of s . By re-arranging, we can see that $s_i = a\theta^i + (a\theta)^2 + (a\theta)^4 + \dots = \text{Tr}(a\theta^i)$. But then

$$(s_0, s_1, \dots, s_{n-1}) = (\text{Tr}(a), \text{Tr}(a\theta), \dots, \text{Tr}(a\theta^{n-1}))$$

and more generally:

$$(s_i, s_{i+1}, \dots, s_{i+n-1}) = (\text{Tr}(\beta), \text{Tr}(\beta\theta), \dots, \text{Tr}(\beta\theta^{n-1}))$$

where $\beta = a\theta^i$, and so the subsequences of length n form an irreducible cyclic code of length n , \mathcal{C} , and the roots of f , θ , are the roots of the irreducible polynomial that defines \mathcal{C} . This implies that f is the reciprocal of the check polynomial of \mathcal{C} as required. \square

Note that the choice of initial terms in the sequence corresponds to the choice of β in the definition of irreducible cyclic codes.

We will now go on to show that this code corresponds to a minimal cyclic ideal.

Lemma 2.42. The code \mathcal{C} with generator polynomial equal to $g = (x^n - 1)/f$ for irreducible f corresponds to a minimal ideal generated by f .

Proof. By converting from codewords to polynomials, we can see that \mathcal{C} is represented by an ideal generated by g :

$$\mathcal{C} = \langle g \rangle = \{gh : h \in \text{GF}(2)[x]\} \pmod{x^n - 1}$$

Assume that this is not minimal, that is, there exists some $g' \neq 0$ such that $\langle g' \rangle \subsetneq \langle g \rangle$. Then $g' \in \langle g \rangle$ and so g divides g' , but if g' generates an ideal then g' divides $x^n - 1$. By the definition, we also know that $g = (x^n - 1)/f$, and f is irreducible, so $g' = x^n - 1$. But then $g' \equiv 0 \pmod{x^n - 1}$, which is a contradiction, and so $\langle g \rangle$ must be minimal. \square

From now on we will refer to irreducible cyclic codes (rather than minimal cyclic codes) as this appears to be the more common term in the literature.

Chapter 3

Literature Review

3.1 Linear Complexity

To determine the security of a sequence that is to be used in a cryptographic protocol, it is crucial to be able to determine the linear complexity of the sequence or at least establish a lower bound on its value. Furthermore, from a cryptanalysis viewpoint, it is desirable to be able to determine the remainder of a sequence if only part of it is known, which is possible once a characteristic polynomial has been found. Determining the remainder of the sequence is easiest when the characteristic polynomial is as small as possible. Since determining the minimal characteristic polynomial gives us the linear complexity of a sequence for free, it is easy to see why being able to find the minimal polynomial is regarded as one of the most crucial aspects to the security of sequences. It is this property which we will focus on.

It is possible to find the smallest characteristic polynomial which can generate the sequence simply by trying all relations, starting with the shortest ones, but this is a highly inefficient process. A polynomial time algorithm is possible by solving a system of linear equations, which runs in $O(n^3)$ operations. The work of Berlekamp [15] on BCH codes was adapted to the problem of determining the characteristic polynomial of a sequence by Massey [47] and this resulted in a superior algorithm which finds the shortest relation in $O(n^2)$ operations. Furthermore, the Berlekamp-Massey algorithm determines the characteristic polynomial from only $2n$ terms (where n is the linear complexity), which is a small part of the period, as the period could be up to $p^n - 1$ terms long (where p is the characteristic of the underlying field).

The Berlekamp-Massey algorithm is still the best known algorithm for determining the linear complexity of a general sequence, in terms of both its computational complexity, and the number of terms of the sequence needed before the char-

acteristic polynomial can be determined. However, for certain classes of sequences, improved algorithms have been found. The Games-Chan algorithm [28] takes linear time and works for binary sequences with period of the form 2^n . It exploits the fact that in this case the minimal polynomial is a factor of $x^{2^n} - 1 = (x - 1)^{2^n}$; hence it is a power of $x - 1$ and it is only needed to determine which power. The Games-Chan algorithm assumes that a whole (not necessarily minimal) period of the sequence is known.

The Games-Chan algorithm has been generalised to fields of arbitrary characteristic by Ding *et al.* in [22]. It too has linear computational complexity, and requires a full period of the sequence to be known for the characteristic polynomial to be determined. It was also generalised to binary sequences of period $u2^v$ (for odd u) by Meidl [53].

It was noted by Sălăgean in [64] and by Meidl in [54] that it is actually not necessary to have a whole period of the sequence in order to determine its linear complexity using the Games-Chan algorithm. It suffices to have a number of terms greater or equal to the linear complexity, provided we still know that the sequence admits as a characteristic polynomial a power of $x - 1$ or more generally of some irreducible polynomial f . For finite sequences which have a characteristic polynomial of the form f^v Meidl gives two algorithms in [54]: one for $f = x - 1$ and arbitrary v , the other for arbitrary f and v being a power of 2. These algorithms both have linear time computational complexity.

Other sequences have also been studied to determine their linear complexity, for example sextic residue sequences were studied by Kim and Song [40], who were able to find a closed form expression for the linear complexity of such sequences, dependent only on their period.

3.2 k -Error Linear Complexity

The notion of k -error linear complexity was first proposed by Diffie in 1989 [27]. By using the Berlekamp-Massey algorithm, on any sequence that differs from the original sequence in no more than k positions, it is possible to determine the k -error linear complexity. However, the computational complexity of this algorithm is superpolynomial - specifically it is: $\sum_{i=0}^k \binom{m}{i} (p - 1)^i$ for a sequence of length m over a field of size p . This approach, which is known as the exhaustive search method, is not considered practical due to its high computational requirements and the need to know an entire period of the sequence. Currently there is no known polynomial time algorithm for finding the k -error linear complexity of a general sequence, although methods are known that are faster than the exhaustive search method, and in some special cases, a polynomial time algorithm is known.

The algorithm of Stamp and Martin [69] is based upon the algorithm of Games and Chan, and introduces errors into the sequence to reduce the running total of linear complexity that is calculated by the Games-Chan algorithm. Just as with the Games-Chan algorithm, an entire period of the sequence has to be known, and that period length must be a power of two and just as with the Games-Chan algorithm its computational complexity is linear in the period of the sequence.

The algorithm of Lauder and Paterson [43] builds upon the Stamp-Martin algorithm and calculates the full linear complexity spectrum without having to determine the k -error linear complexity for each value of k . Repeatedly applying the Stamp-Martin algorithm, would have a computational complexity of $\mathcal{O}(N^2 \log N)$, but the improved algorithm of Lauder and Paterson, reduces this to $\mathcal{O}(N(\log N)^2)$. Lauder and Paterson also presented an algorithm for determining the specific sequence which has the lowest linear complexity out of all those at distance up to k from the given sequence, which is not explicit from the algorithm of Stamp and Martin. The work of Stamp and Martin was extended by Zhou [74] to sequences over a field of any prime size, and whose period length is $2p^n$. Further results on the linear complexity spectrum were presented by Etzion *et al.* [26] where upper and lower bounds on the number of critical points the linear complexity spectrum can have were established.

The biggest strength of the algorithm of Berlekamp and Massey from a cryptographic viewpoint is that it can find the linear complexity, n , of a sequence given just $2n$ terms. However the algorithms of Stamp and Martin and Lauder and Paterson require knowledge of the full sequence before they can be applied. The work of Sălăgean [64] showed that the Games-Chan algorithm can be adapted so that it only requires knowledge of $2n$ terms to determine the linear complexity. As the Stamp-Martin and Lauder-Paterson algorithms are based on the algorithm of Games and Chan, adaptations to these algorithms were also presented in [64] so that they too only require knowledge of $2n$ terms of the sequence, where n is the linear complexity of the original sequence.

The results of Kaida, Uehara and Imamura [39] extended upon the work of Lauder and Paterson by generalizing their methods from sequences over fields with characteristic 2 to fields with arbitrary prime characteristic.

Whilst the algorithms mentioned above impose a condition on the length of the sequence, Meidl [54] instead imposes a condition on the form of the characteristic polynomial of the sequence. That is, an algorithm is presented that can determine the k -error linear complexity of a binary sequence whose characteristic polynomial is a power of $x^2 - x - 1$. Note that any condition on the length of the period of a sequence is equivalent to a condition on the characteristic polynomial as a sequence has a period of length N exactly when $x^N - 1$ is a characteristic polynomial.

3.3 Approximation Algorithms

An entirely different approach to finding the k -error linear complexity was proposed by Alecu and Sălăgean [2]. An evolutionary computation model is presented, whereby a large number of randomly selected error sequences are initially chosen, and then through an evolutionary process it is attempted to find the optimal solution. It is shown that each of the commonly used techniques for modeling evolutionary processes produces highly similar results, and that these results were, in all cases, close to the actual values of the k -error linear complexity. However, it is not possible to find the exact value of the k -error linear complexity using this approach, and furthermore, the linear complexity of the algorithm is super-polynomial in the linear complexity of the sequence.

Another alternative approach to the problem of determining k -error linear complexity was proposed by Alecu and Sălăgean [3] [4] [66]. Their results extended the earlier work of Blackburn [16], Paterson [61], Massey and Serconek [49] and Massey [48], who each used Blahut's Theorem to determine the linear complexity of a sequence by transforming the sequence using the Discrete Fourier Transform. The algorithm presented by Alecu and Sălăgean has polynomial computational complexity, but is only an approximation algorithm, and does not guarantee to find the actual k -error linear complexity.

3.4 General Results on k -Error Linear Complexity

Although very little is known on the distribution of the k -error linear complexity in general, there are some classes of sequence, for which closed form expressions on the k -error linear complexity are known. Meidl and Niederreiter [55] present lower bounds for the k -error linear complexity of sequences over $\text{GF}(q)$ with period of the form $n = p^v$ or $n = p^v r$ where p is the characteristic of the field $\text{GF}(q)$ and r is a prime. In some specific circumstances, these lower bounds were improved to equalities. These results are based on knowledge of the number of sequences of length N with a particular linear complexity.

The work of Aly and Winterhof [6] developed formulae for the k -error linear complexity of two specific families of sequences, Legendre sequences and Sidelnikov sequences. For these results, no knowledge is required other than the field over which the sequence lies, and the sequence itself. Further work was later carried out by Aly, Meidl and Winterhof on cyclotomic sequences [51]. They found lower bounds for the k -error linear complexity of such sequences, and in certain

instances, equalities. These results were of particular interest as the k -error linear complexity of such sequences was shown to be high in all cases, which implies that these sequences may be suitable for use in cryptographic applications.

Another, closely related, problem was studied by Meidl and Niederreiter [56] [58] who found a class of sequences which simultaneously achieve the maximal linear and k -error linear complexity. Before this work it was not known whether sequences with such properties existed, and so this was a crucial work on proving that sequences can be found that are arbitrarily secure in relation to both of these measures. Continuing on from this, Meidl [52] derived an exact formula for determining the k -error linear complexity for sequences having maximal possible linear complexity.

3.5 General k -Error Knowledge

An alternative, but closely related problem, to that of determining the k -error linear complexity of a sequence, is to study the effects on the linear complexity of sequences in general, when certain terms, or combinations of terms are altered.

Fell [27] determined the average change in linear complexity when a term early in a finite sequence is changed, as well as the average change in linear complexity when the final term is changed, and determined that changing bits at the beginning of the sequence was much less likely to change the linear complexity. Also, results were presented on the impact of an arbitrary bijection on the terms of a period of a sequence. Permuting the terms of a sequence is equivalent to changing some of them, with the added restriction that the total number of terms of any particular value in the field must be unchanged. An upper bound was established on the average change in linear complexity, which was dependent on the exact length of the period, but lower than $5/3$, independent of period.

The work also presents results on the theoretical capabilities of any k -error algorithm. By considering a k -error algorithm as a mapping from any given sequence to one with lower linear complexity, it is shown that a low value of k will ensure that there are certain sequences whose linear complexity can only be reduced by a small amount. Similarly, if it is required that every sequence have its linear complexity reduced to a low value, then the value of k must necessarily be large.

Meidl and Niederreiter [55] presented results on the total number of sequences of a given length whose k -error linear complexity was below a certain threshold. Specifically, the number of sequences of length N , whose k -error linear complexity was equal to 0, or less than 1 or $N - 1$ was presented. These results are used to calculate the expected value of the k -error linear complexity for sequences with

prime period, and lower bounds on the k -error linear complexity in a slightly more general case.

This work was later extended by Meidl [54], to the number of sequences with period length a power of 2 whose 1- or 2-error linear complexity is strictly less than the linear complexity of the sequence itself. This is used to determine the expected value for the 1- or 2-error linear complexity, for sequences with period length a power of 2.

Xiao, Wei, Lam and Imamura [71] presented results on the linear complexity of sequences over $\text{GF}(q)$, where q is a prime, with period length p^n where p is an odd prime and q is a primitive root modulo p^2 . Results from number theory on the properties of cyclotomic polynomials were used to develop an algorithm for determining the linear complexity of a sequence which has linear computational complexity. These results were extended by Han, Chung and Yang [32] to results concerning k -error linear complexity. A value of k is found, such that for any sequence the k -error linear complexity is strictly less than the linear complexity. Results are also presented on the expected value of the k -error linear complexity for certain values of k , and upper and lower bounds on the k -error linear complexity for certain values of k .

3.6 Determining the Required Number of Errors

Most of the results mentioned above are concerned with the situation where a fixed number of errors are introduced with the goal of altering the linear complexity. Kurosawa, Sato, Sakata and Kishimoto [42] presented results on the converse problem, that is, what is the minimum number of changes that need to be made to a sequence to cause a specified change in the linear complexity. Specifically, they provided results on the minimum number of errors needed to strictly reduce the linear complexity for a sequence whose period length is a power of 2, and also results on the new value of the linear complexity in these cases.

This work was extended by Sălăgean [64] who presented an algorithm for finding the minimum number of errors required to reduce the linear complexity by any amount, in linear time. As with the work by Kurosawa et al. the results are only applicable to binary sequences of period a power of two.

3.7 Feedback Carry Shift Registers

In this section we will refer only to the binary situation for ease of notation, but all the concepts can be applied easily to larger base fields. Klapper and Goresky [41] introduced the notion of Feedback Carry Shift Registers (FCSRs) as an alternative to Linear Feedback Shift Registers (LFSRs). The model of the FCSR is similar to that of the LFSR, but contains an extra register. This register is known as the memory cell, and (unlike the other cells) is capable of taking any integer value. Let M be the contents of the memory cell, a_i the characteristic constants, and s_i the contents of the n registers. Then the characteristic function for the FCSR is $\sum_{i=0}^n a_i s_i + M = \sigma$. The register is updated by entering $\sigma \pmod{2}$ as s_n , and the memory register is updated to be $(\sigma - s_n)/2$, and all other registers are updated by $s_i \leftarrow s_{i+1}$.

The FCSR model allows us to assign another value of linear complexity to a sequence, namely the length of the shortest FCSR needed to generate it, plus the size of the additional memory cell. Since the memory cell can take any value in the integers, it is not of a fixed size, and so this is why it must be included in the complexity of a register. This measure is referred to as the 2-adic complexity of the sequence, since the analysis of FCSRs relies heavily upon 2-adic arithmetic. The formulation of a new measure of complexity also allows us to define a new type of k -error complexity, referred to as the k -Error 2-Adic Complexity which was originally proposed by Hu and Feng [36]. In their work they propose some lower bounds for this value, but they do not improve upon the analogous values known for the standard k -error linear complexity.

3.8 Irreducible Cyclic Codes

Determining the weight enumerator polynomial of an irreducible cyclic code is a problem that is closely related to determining the k -error linear complexity of a sequence, see Chapter 6. An overview of the current state of knowledge in this area is given by Ding [23], which shows that it is still an open problem to determine the weight enumerator polynomial for a general sequence. However, for certain classes of sequence, the weight enumerator polynomial is known.

The first results in this area were given by McEliece and Rumsey [50], who presented a way of calculating the weight enumerator polynomial for a certain class of codes, namely those that correspond to cyclotomic sequences. This result was extended by Baumert and McEliece [11] who were able to determine the weight enumerator polynomial for all codes in a class known as semi-primitive. For more details on these codes see Chapter 7. Another key result in this area was

presented by Baumert and Mykkeltveit [12] who determined the weight enumerator polynomial for another class of irreducible codes, namely those which satisfied a certain condition on the length of the codes. Again, for more details, see Chapter 7. Hellesteth, Kløve and Mykkeltveit [34] studied another class of codes, again, which are classified by a condition on their length, and they determined the weight polynomial for these codes.

Another result was presented by Schmidt and White [67] who limited their consideration to two-weight codes. As well as the already known two weight codes, such as the semi-primitive ones, they claimed to have found all further two weight codes, and determined their weight, although they were unable to prove conclusively that they had indeed found all such codes.

Aubry and Langevin [8] studied binary irreducible codes, and were able to determine new results relating to the divisibility of the weights of the codewords, although they were not able to explicitly determine the weights of the codewords themselves.

Although defined differently, minimal cyclic codes are entirely equivalent to irreducible cyclic codes. Research into their weights are therefore also useful for determining the k -error linear complexity of certain sequences. Pruthi and Arora [63] were able to determine the weight enumerator for codes of prime power length and then in [7] extended these results to codes of length $2p^n$ for some odd prime p , and were able to determine the weights of the codewords as well. Bakshi and Raka [10] investigated codes of length p^nq for some primes p and q and were able to find the minimal distance of such codes, which is useful, but does not provide as much information as the weight enumerator polynomial.

An alternative approach had been pursued by MacWilliams and Seery [46], who instead of trying to find algorithms or expressions for certain classes of codes, chose to compute the weights of irreducible codes for all instances of codes up to length $2^{27} - 1$.

3.9 Stream Cipher Proposals

The largest single contribution to the field of stream cipher proposals in recent years has been the eStream project. For a full list of schemes that were proposed and studied, please see the website of the project [25]. The full details of the various proposals and the attacks that have been carried out on them is beyond the scope of this thesis, but we will mention the ciphers which exhibit properties that make the results presented in this thesis particularly applicable.

Decim (and its variants) [13] is a hardware orientated stream cipher, which uses an LFSR to construct an m-sequence of period length $2^{192} - 1$.

Grain [33] is another hardware targeted stream cipher, which also uses an LFSR to generate an m-sequence. The sequence in this case has period length $2^{80} - 1$.

Pomaranch [38] is a stream cipher based on jump registers and LFSRs. Due to the nature of the jump registers and the way the LFSR is irregularly clocked, it is not possible to know, in advance, the period length of the outputted sequence.

Trivium is also based on three separate NLFSRs, and the input to each is dependent on the other two, in a circular dependency fashion. Therefore, again, it is not possible to determine the exact value of the period length of the output.

MICKEY [9] uses two registers, one linear and one non-linear. However, even though the feedback polynomial for the LFSR is known, the register is also dependent on the output of the NLFSR and so the LFSR cannot be analysed in isolation.

SOSEMANUK [14] is a software focused cipher, which uses an LFSR as part of its design. However, unlike most other ciphers, it employs a cipher over $\text{GF}(2^{32})$, instead of over $\text{GF}(2)$. This is done as storing 32-bit words in software is very easy (as is carrying out operations on them), and doing so increases the speed of the cipher. Similarly SOBER [31] (which was not an eStream Candidate) uses an LFSR over $\text{GF}(2^8)$.

Outside of the eStream project, there have been many individual proposals, far too many to list here. We will just mention two that are particularly relevant to this thesis:

SSC2 [73] uses an LFSR to generate an msequence of length $2^{128} - 1$ as part of its design, and LILI-128 [21] uses two LFSRs both of which generate m-sequences, of length $2^{39} - 1$ and $2^{89} - 1$.

3.10 Contribution to Knowledge

In this field there are many open problems that have been discussed above and in this thesis we will investigate some of those. Firstly, there is the problem that there is no known polynomial time algorithm for determining the k -error linear complexity of a general sequence, either finite or infinite periodic. There is also no known way of calculating the k -error period of a general sequence in polynomial time. Whilst there are algorithms for determining the linear complexity of a general sequence, it is desirable to be able to reduce the time complexity of the algorithm for certain classes of sequence.

We first present a heuristic algorithm for determining the k -error linear complexity of a sequence. We present versions of this algorithm for both infinite and finite sequences. The algorithm is based on a technique whereby it is assumed that

the correct minimal polynomial has been found, by using a bi-directional variation of the Berlekamp-Massey algorithm, and then individual errors are located at the appropriate locations throughout the sequence. We determine the situation in which the algorithm can be guaranteed to find the correct k -error linear complexity, and determine the proportion of sequences which possess this property out of all sequences of a given length. This proportion is exponential in the length of the sequence, and yet the complexity of the algorithm is polynomial. We also present results on the performance of the algorithm on the sequences on which it is not guaranteed to find the correct k -error linear complexity. We then present a variation of this algorithm for the situations whereby different assumptions can be made, namely that a full period of the sequence is known.

We then go on to present an algorithm for determining the linear complexity of a certain class of sequences, namely those with characteristic polynomial a power of an irreducible polynomial. The key advantage of this algorithm is that its computational complexity is linear in the length of the sequence. Our algorithm generalises a number of previously known algorithms, namely those of Games and Chan (and variations of it), Ding *et al.*, and Meidl. For ease of use, we present variations of this algorithm for the general case, and because of its importance, the case whereby the underlying field has characteristic 2. We then go on to present a variation of this algorithm for the case whereby the input is a finite sequence, generalising the algorithms of Sălăgean and Meidl. Again, this algorithm has linear computational complexity. Finally we present a variation of the algorithm whereby the input sequence has a characteristic polynomial which is the product of known irreducible factors. The complexity of this algorithm depends on the specific set of irreducible polynomials that is used, but in general, the larger and more general the set of sequences, the less efficient the algorithm is. The majority of the results in this chapter were originally published as [19].

We finally go on to study various properties of m -sequences, and their use in stream cipher proposals. We focus our efforts on the k -error linear complexity of m -sequences and the closely related idea of k -error period, which we believe to be a new notion, although it has been looked at implicitly before. We also study the complexity stability threshold and the period stability threshold of m -sequences, which we also believe to be a new notion. We split the problem up into various different cases, some of which we are able to solve completely, some of which we are only able to partially solve, and some of which we are only able to prove very basic results for. However, for virtually all m -sequences, we provide a method of reducing the complexity of the problem, and so for many sequences, although we do not have a closed form expression, we are able to apply a small amount of brute force computation to find the required results. We also apply our results to several

stream cipher proposals, and are able to show that from a stability viewpoint some of the sequences used are more secure than others, although all performed very well. We finally use some further results from coding theory to study more classes of m-sequence. The results of Chapter 6 were originally published in [20].

Although all of the core results that are presented here were originally developed by myself, they were developed and refined through an iterative process with my main supervisor (Dr. Sălăgean), and so the credit for the final form of these results must be partly split between us.

Chapter 4

A Heuristic Algorithm for Determining k -error Linear Complexity of a Sequence

4.1 Introduction

In this chapter we will introduce a heuristic algorithm for approximating the k -error linear complexity of either a finite or infinite periodic sequence. The algorithm is based on the idea of viewing the error sequence as a series of individual errors, rather than as one complete sequence. We provide some results as to the size of the search space that is covered by the algorithm, and to the average behavior of the results obtained. We also provide a variant of the algorithm, which runs on a slightly different input type, and again provide results on its theoretical and observed performance.

4.2 The Algorithm

Finding a sequence s' within Hamming distance k of s is equivalent to finding an *error sequence*, e , with Hamming weight less than k , such that $s + e = s'$ (where the addition is carried out termwise). In this chapter we will analyse the sequence term-by-term, rather than analyzing it as a whole. Therefore, the problem is transformed from one of trying to find an optimal error sequence to one of trying to find the optimal locations for the individual errors to be placed. Once this has been done, it is trivial to construct the optimal error sequence.

The algorithm proposed here uses the Berlekamp-Massey algorithm repeatedly by making the assumption that it has the correct characteristic polynomial for the sequence, and then uses this relation to locate errors where the sequence disagrees

with the value predicted by the polynomial. This idea is similar to the one used by Stamp and Martin [69], but is used in a much more general setting here. In this way the algorithm passes along the length of the sequence, locating errors where the sequence disagrees with the predicted values. There are then two situations which can cause this process to terminate: either the algorithm requires more than k errors, or the algorithm reaches the end of the sequence without using too many errors, and so has found a candidate for the error sequence.

The recurrence relation is initially generated by taking a number of terms of the sequence, the *initial segment*, and by using the Berlekamp-Massey algorithm to find a relation that generates this part of the sequence. Initially the shortest such segments are used, and are taken starting at the first term of the sequence. To maximise the probability of finding an optimal solution by maximising the number of proposed relations the algorithm considers initial segments of varying length and position within the sequence. However this requires the generated relation to be able to generate terms in both increasing and decreasing order of index, a criterion that is not fulfilled by the original Berlekamp-Massey algorithm. Therefore a modified version of the Berlekamp-Massey algorithm is used, that fulfills this requirement, known as bi-directional, developed by Sălăgean [65].

Once all possible segments in the sequence of a certain length have been tried as the initial segment, it is necessary to use longer segments, and so generate longer recurrence relations. Then all possible segments with this length must be examined and the process repeated until a relation is found that does not require more than k errors to generate the entire sequence. The algorithm is given in pseudo-code form as *kErrAp*.

4.3 Notes on the Algorithm

There are several points in the algorithm *kErrAp* which should be explained. The line labeled (11) will determine the discrepancy between the predicted value for the next term in the sequence, and the actual value. The two similar while loops (lines (10) to (21) and (23) to (34)) analyse the sequence in opposite directions.

It may appear that once the algorithm has found a success, that this must be the smallest such relation, and that the algorithm can terminate. This would have the advantage of reducing the expected running time of the algorithm, but may, in certain circumstances, lead to the algorithm missing a shorter relation. This is because the algorithm does not search through relations in order of increasing linear complexity, but searches through initial segments of the sequence in order of increasing length. While this will usually lead to a longer relation, this is not always the case, and so it is important to continue to run the algorithm. An

Algorithm 1: *kErAp*

Input : A sequence S , the length of S , n , and an integer, k
Output: The approximate k -error linear complexity of S

```

1 begin
2   for  $L = 1$  to  $n$  do
3     for  $j = 0$  to  $n - L$  do
4        $K \leftarrow 0$ ;
5        $s \leftarrow S$       ( $s$  is initialized as a copy of  $S$ );
6        $Berl \leftarrow BiBM(s, L, j)$ ;
7       ( $BiBM$  is a subroutine which returns the shortest bi-directional
8         recurrence relation on  $L$  terms of  $s$ , starting at  $j$ , where  $Berl[1]$ 
9         is returned as the linear complexity of the relation, and  $Berl[2]$ 
10        are the coefficients of the polynomial)
11       $i \leftarrow j + 1$ ;
12       $e \leftarrow [0, \dots, 0]$ ;
13      while  $K \leq k$  and  $i \leq (n - Berl[1])$  do
14         $d \leftarrow \sum_{i'=0}^{Berl[1]} Berl[2][1 + i'] * s[i + i']$ ;
15        if  $d = 0$  then
16           $e[i + Berl[1]] \leftarrow 0$ ;
17           $i \leftarrow i + 1$ ;
18        else
19           $s[i + Berl[1]] \leftarrow s[i + Berl[1]] - d$ ;
20           $e[i + Berl[1]] \leftarrow -d$ ;
21           $i \leftarrow i + 1$ ;
22           $K \leftarrow K + 1$ ;
23        end
24      end
25       $i \leftarrow j$ ;
26      while  $K \leq k$  and  $i \geq 1$  do
27         $d \leftarrow \sum_{i'=0}^{Berl[1]} Berl[2][1 + i'] * s[i + i']$ ;
28        if  $d = 0$  then
29           $e[i] \leftarrow 0$ ;
30           $i \leftarrow i - 1$ ;
31        else
32           $s[i] \leftarrow s[i] - d$ ;
33           $e[i] \leftarrow -d$ ;
34           $i \leftarrow i - 1$ ;
35           $K \leftarrow K + 1$ ;
36        end
37      end
38      if  $K \leq k$  then
39         $Result[Length(Result) + 1] \leftarrow Berl[1]$ ;
40      end
41    end
42  end
43  Return(Min(Result));
44 end

```

example is given below:

Example 4.1. Consider the sequence $s = 011111110101100010$, over $\text{GF}(2)$ and let $k = 3$. By an exhaustive search we can find that the 3-error linear complexity of s is 5. However when $k\text{ErAp}$ is run on s , the first success it finds is when $L = 8$ and the starting point for the initial segment is the second term. On this initial segment, the Berlekamp-Massey algorithm returns the polynomial $x^7 + x + 1$, which, when used with error sequence 000000000000110001 , generates s . However if the algorithm is allowed to continue, when $L = 10$ and the initial segment is taken starting at the eighth term, then the Berlekamp-Massey algorithm returns the polynomial $x^6 + x^4 + 1$, which has lower degree. This polynomial, when combined with the error sequence 01000100000000001 , also generates s , and so we have seen that by allowing the algorithm to continue to run on longer sequences, it is possible to reduce the linear complexity. It should be noted that there were other successful relations returned for $L = 8$ and $L = 9$, but none of length 6. It should also be noted that although the returned linear complexity is reduced by increasing the length of the initial sequence, it is still not the optimal linear complexity, which is not found by the algorithm in this instance.

We should also mention that there is not necessarily a unique minimal bi-directional relation on the initial sequence. That is, there could be more than one bi-directional relation of minimum length that will generate the terms of the initial segment that is used in $BiBM$. The subroutine, however, will only return one of these, chosen at random. This might appear to mean that a potential relation could be missed by the algorithm, but this is not actually the case, since if one of the unused alternative relations was a suitable candidate, then it will be generated uniquely when the length of the initial segment has been increased until it is twice the length of the relation, with the same initial segment starting point.

4.4 Performance of the Algorithm

Theorem 4.2. If a sequence, s , of length N , has k -error linear complexity n , $2n \leq N$, and the optimal error sequence, e , has a gap of length at least $2n$ (that is, $2n$ consecutive zeros) then $k\text{ErAp}$ will return the k -error linear complexity and an optimal error sequence.

Proof. If the optimal relation is generated at some point during the algorithm, then it will be returned. Therefore it suffices to show that the optimal relation will be generated at some point in the running of the algorithm. Denote this relation as R .

Consider the case when the initial segment to be considered is of length $2n$ and begins at the term which is also the first term of the gap in the error sequence. Then *BiBM* will return R , and the algorithm will only be required to place errors in the optimal positions, and so will consider R as a potential solution. \square

Lemma 4.3. *Let f_k be defined as follows:*

$$f_k(N, n) = \begin{cases} 1 & k = 0 \\ (\min(N, 2N - 4n)) (q - 1) & k = 1 \\ \sum_{i=2n}^{N-k} ((N - 1 - i) \binom{N-2-i}{k-2} + 2 \binom{N-1-i}{k-1}) (q - 1)^k & k \geq 2 \end{cases}$$

Consider the set of sequences of length N over $\text{GF}(q)$. Then when $N/2 \leq 2n < N$ the number of sequences whose optimal error sequence has a gap of size at least $2n$ is given by

$$\sum_{i=0}^k f_i(N, n)$$

Proof. In Lemma 4.2 it was shown that *kErAp* will find the optimal error sequence whenever such a sequence has a sufficiently large gap. The condition of $2n \geq N/2$ is necessary to ensure that the gap we are considering is the largest such gap, otherwise the function will double count some error sequences. The three cases given for f_k are considered separately.

When $k = 0$ the k -error linear complexity of the sequence is equal to the linear complexity. There is only one possible error sequence in this case, the all zero sequence, and the algorithm will run until $L = 2n$ at which point the *BiBM* subroutine will return the optimal recurrence relation and the algorithm will terminate.

When $k = 1$ the error gap will be large enough if the single error occurs at either the beginning or the end of the sequence. More specifically it will be large enough if the error occurs in position $1, 2, 3, \dots, N - 2n$ or $2n + 1, 2n + 2, \dots, N$. If $N - 2n < 2n + 1$ then there is no overlap between these two sections, but if there is such an overlap, then the error can occur at any position in the sequence, namely at any one of the N positions. If $N - 2n < 2n + 1$ holds then there are $N - 2n + (N - 2n) = 2N - 4n$ possibilities. For each of these error positions, there are $q - 1$ choices for the value of the error, since it can take any value other than zero.

Finally consider the case when $k \geq 2$. The error-sequences that are found by the algorithm can be classified by the exact length of their largest gap, and the position of this gap. The gap lengths can range from $2n$ to $N - k$ giving $N - k - 2n + 1$ different lengths. We now split such sequences into two groups, those where both ends of the gap are denoted by a non-zero element, or those

where one end of the gap is denoted by the end of the sequence. In the first case, we can place the gap anywhere apart from at the two ends, so if we assume that the length of the gap is i (where $2n \leq i \leq N - k$) then there are $N - 1 - i$ positions for this gap. Then the remaining $k - 2$ errors can be placed in any one of the remaining $N - 2 - i$ positions, giving the first term in the sum. The second case considers when one end of the gap is given by the end of the sequence, and in this case, regardless of the length of the gap, there are only ever 2 valid positions for it. Once the position has been fixed the remaining $k - 1$ errors can be placed in any of the remaining $N - 1 - i$ terms. This gives the second term in the sum. For each such error sequence, each of the k errors can take any one of the $q - 1$ non-zero values in $\text{GF}(2)$, and so we need to consider each set of error locations $(q - 1)^k$ times.

We have now seen that f_k gives us the number of errors for each specific value of k , and it just remains to note that for the k -error linear complexity it is necessary to check all error sequences with the number of errors less than or equal to k , and this gives the final sum. \square

Corollary 4.4. *If f_k is defined as in Lemma 4.3 then the number of error sequences considered by $k\text{ErAp}$ is given by*

$$\sum_{i=0}^k f_i(N, n)$$

That is, $k\text{ErAp}$ will return the correct k -error linear complexity for a sequence if its optimal error sequence is of the right form, and the set of sequences of the right form is of exponential size.

Remark 4.5. *It should be noted that the fraction of the error sequence space considered by the algorithm does not necessarily relate to a probability of success. This is for two reasons. Firstly, the error sequences may not be distributed uniformly over this space, and secondly, each sequence may have more than one optimal recurrence relation, and so more than one error sequence. Since only one such error sequence needs to be in the form given above, the chances of finding a successful error sequence will be improved.*

Corollary 4.6. *The number of error sequences considered by $k\text{ErAp}$ is exponential.*

Theorem 4.7. *Algorithm $k\text{ErAp}$ has running time of $\mathcal{O}(N^4)$, where N is the length of the sequence.*

Proof. $k\text{ErAp}$ consists of 2 main for loops. Inside the inner most loop there is a call to the bi-directional Berlekamp-Massey subroutine, and two while loops.

The number of executions of each while loop depends on the value of j , but between them, the number of executions is constant, namely $N - Berl[1]$, where $Berl[1]$ is the linear complexity returned by the subroutine. The complexity of the subroutine itself is $\mathcal{O}((2L)^2)$ [64]. This means that for each iteration of the innermost for loop, the complexity will be $\mathcal{O}(((2L)^2) + N) = \mathcal{O}(N^2)$. The inner for loop is run up to $(N - 1)$ times, and the outer for loop is run up to N times. This gives a total complexity of $\mathcal{O}(N * N * N^2) = \mathcal{O}(N^4)$ \square

Example 4.8. *In the following example we outline the running of $kErAp$ for the example of the sequence 000100110111, for when $k = 2$, and the sequence is taken over $\text{GF}(2)$. We give the values of L and j , the characteristic polynomial returned by the subroutine, the number of errors that have been used after each of the while loops, and finally whether or not the algorithm has found a candidate for the optimal linear complexity. If the number of errors used has exceeded 2, then we give the last position in which an error was placed (given in brackets). The best solution found by the algorithm is $x^3 + x^2 + x + 1$, which is found on two separate occasions, namely $L = 5, j = 4$ and $L = 6, j = 3$.*

L	j	Polynomial	Errors After		Success
			First While	Second While	
			(Location of third error)		
2	0	1	3(8)	-	
	1	1	3(8)	-	
	2	$x^2 + 1$	3(9)	-	
	3	$x^2 + 1$	3(9)	-	
	4	1	3(10)	-	
	5	$x^2 + 1$	3(9)	-	
	6	$x^2 + 1$	1	3(5)	
	7	$x^2 + 1$	1	3(6)	
	8	$x^2 + 1$	1	3(6)	
	9	$x^2 + 1$	0	3(5)	
10	$x^2 + 1$	0	3(5)		
3	0	1	3(8)	-	
	1	$x^3 + 1$	3(9)	-	
	2	$x^2 + 1$	3(9)	-	
	3	$x^3 + 1$	3(9)	-	
	4	$x^3 + 1$	3(9)	-	
	5	$x^2 + x + 1$	1	3(2)	
	6	$x^2 + x + 1$	1	3(2)	
<i>continued on next page</i>					

L	j	Polynomial	Errors After		Success
			First While	Second While	
			(Location of third error)		
	7	$x^2 + 1$	1	3(6)	
	8	$x^2 + x + 1$	1	3(2)	
	9	$x + 1$	0	3(5)	
4	0	$x^4 + 1$	3(7)	-	
	1	$x^3 + 1$	3(9)	-	
	2	$x^3 + 1$	3(9)	-	
	3	$x^3 + 1$	3(9)	-	
	4	$x^3 + x + 1$	3(9)	-	
	5	$x^2 + x + 1$	1	3(2)	
	6	$x^2 + x + 1$	1	3(2)	
	7	$x^2 + x + 1$	1	3(2)	
8	$x^3 + x + 1$	0	3(3)		
5	0	$x^4 + 1$	3(7)	-	
	1	$x^3 + 1$	3(9)	-	
	2	$x^3 + 1$	3(9)	-	
	3	$x^3 + x + 1$	3(9)	-	
	4	$x^3 + x^2 + x + 1$	1	2	Yes
	5	$x^2 + x + 1$	1	3(2)	
	6	$x^2 + x + 1$	1	3(2)	
7	$x^3 + x^2 + 1$	0	3(3)		
6	0	$x^4 + 1$	3(7)	-	
	1	$x^3 + 1$	3(9)	-	
	2	$x^3 + x + 1$	3(9)	-	
	3	$x^3 + x^2 + x + 1$	1	2	Yes
	4	$x^4 + x^2 + x + 1$	2	3(4)	
	5	$x^2 + x + 1$	1	3(2)	
6	$x^4 + x^3 + x^2 + x + 1$	0	3(4)		
7	0	$x^4 + x^3 + 1$	1	1	Yes
	1	$x^4 + x^3 + 1$	1	1	Yes
	2	$x^4 + x^3 + 1$	1	1	Yes
	3	$x^4 + x^3 + 1$	1	1	Yes
	4	$x^4 + x^2 + x + 1$	0	2	Yes
5	$x^4 + x^2 + x + 1$	0	2	Yes	
8	0	$x^4 + x^3 + 1$	1	1	Yes

continued on next page

L	j	Polynomial	Errors After		Success
			First While	Second While	
	1	$x^4 + x^3 + 1$	1	1	Yes
	2	$x^4 + x^3 + 1$	1	1	Yes
	3	$x^5 + x^4 + x^3 + x + 1$	0	2	Yes
	4	$x^5 + x^2 + x + 1$	0	2	Yes
9	0	$x^4 + x^3 + 1$	1	1	Yes
	1	$x^4 + x^3 + 1$	1	1	Yes
	2	$x^5 + x^4 + x^2 + 1$	0	1	Yes
	3	$x^5 + x^4 + x^3 + x + 1$	0	2	Yes
10	0	$x^4 + x^3 + 1$	1	1	Yes
	1	$x^6 + x^4 + 1$	0	1	Yes
	2	$x^5 + x^4 + x^2 + 1$	0	1	Yes
11	0	$x^7 + x^4 + x^3 + 1$	0	0	Yes
	1	$x^6 + x^4 + 1$	0	1	Yes
12	0	$x^7 + x^4 + x^3 + 1$	0	0	Yes

Table 4.1: Example Running of *kErAp*

4.5 Numerical Results

In Tables 4.2, 4.3 and 4.4 we tabulate the size of the space searched by *kErAP* for various values of N , n and k , where N is the sequence length and n is the k -error linear complexity. Each of these examples is considered over $\text{GF}(2)$.

From the tables, the fraction of the total error space that is searched is dependent upon the value of k , and the difference between N and n . However, since there is no way of knowing beforehand what the value of n will be, this makes it difficult to estimate the size of the space that will be searched. A simple upper bound on the complexity can be established by calculating the linear complexity of the sequence, and hence a simple lower bound on the search space can be established, but this is likely to be a very crude measure, especially for higher values of k .

The algorithm does not provide any guarantees about the results provided when the optimal error sequence is not of the required form. Instead we have run the algorithm on a randomly generated set of 50 sequences, each of length 30. We then compared the results with those generated by an exhaustive search algorithm. The results are displayed in Table 4.5.

N=32, k = 1				
	n=8	10	12	14
Total Space Size	33	33	33	33
Searched Space	33	25	17	9
Percentage	100.0%	75.8%	51.2%	27.3%
N=32, k = 2				
	n=8	10	12	14
Total Space Size	529	529	529	529
Searched Space	393	223	101	27
Percentage	74.3%	42.2%	19.1%	5.1%
N=32, k = 3				
	n=8	10	12	14
Total Space Size	5489	5489	5489	5489
Searched Space	2633	1103	325	43
Percentage	48.0%	20.1%	5.9%	0.8%

Table 4.2: Size of the search Space, N=32

4.6 Alternative Implementation

By applying the idea behind the algorithm to a different model, we can produce a different algorithm, with different results. Namely, if we assume that we have a full period of the sequence, then we can produce more promising results. By assuming the full period, we effectively increase the length of the sequence we are studying, as we can assume repetitions of the sequence and so this gives us more opportunity to find a sufficiently large gap in the error sequence, and so find the optimal relation. The adjusted algorithm is presented below as *kErApFP*.

There are certain points in the algorithm which should be explained. Firstly, because we are now assuming a full period, there is no need for the recurrence relation to be forced to be bidirectional, since the repetition of the sequence will naturally do this. Also when an array index is taken (mod n) it is understood that if the index is equal to n , then the n -th term in the array will be taken, rather than the zero-th.

Theorem 4.9. *If a sequence, s , has k -error linear complexity n , and the optimal error sequence, e , has a gap of length at least $2n$, or the leading and trailing zeros in the error sequence create a gap of length $2n$ then *kErApFP* will return the k -error linear complexity and the optimal error sequence.*

Proof. This is simply the analogue of Lemma 4.2, with the added condition that if the gap occurs over the end of the period then the algorithm will also find the optimal sequence. This is clear from the algorithm. \square

N=48, k = 1						
	n=12	14	16	18	20	22
Total Space Size	49	49	49	49	49	49
Searched Space	49	41	33	25	17	9
Percentage	100%	83.7%	67.3%	51.0%	34.7%	18.4%
N=48, k = 2						
	n=12	14	16	18	20	22
Total Space Size	1177	1177	1177	1177	1177	1177
Searched Space	877	611	393	223	101	27
Percentage	74.5%	51.9%	33.4%	18.9%	8.6%	2.3%
N=48, k = 3						
	n=12	14	16	18	20	22
Total Space Size	18770	18770	18770	18770	18770	18770
Searched Space	8973	5171	2633	1103	325	43
Percentage	47.8%	27.5%	14.0%	5.9%	1.7%	0.2%

Table 4.3: Size of the search Space, N=48

N=60, k = 1							
	n=16	18	20	22	24	26	28
Total Space Size	61	61	61	61	61	61	61
Searched Space	57	49	41	33	25	17	9
Percentage	93.4%	80.3%	67.2%	54.1%	41.0%	27.9%	14.8%
N=60, k = 2							
	n=16	18	20	22	24	26	28
Total Space Size	1831	1831	1831	1831	1831	1831	1831
Searched Space	1191	877	611	393	222	101	27
Percentage	65.0%	47.9%	33.4%	21.5%	12.1%	5.5%	1.5%
N=60, k = 3							
	n=16	18	20	22	24	26	28
Total Space Size	36051	36051	36051	36051	36051	36051	36051
Searched Space	14295	8973	5171	2633	1103	325	43
Percentage	39.7%	24.9%	14.3%	7.3%	3.1%	0.9%	0.1%

Table 4.4: Size of the Search Space, N=60

Algorithm 2: *kErApFP*

Input : A sequence S , n , the length of S and an integer k **Output:** The approximate k -error linear complexity of S

```

1 begin
2   for  $L \leftarrow 2$  to  $n$  do
3     for  $j \leftarrow 0$  to  $(n - L)$  do
4        $K \leftarrow 0$ ;
5        $s \leftarrow S$  ( $s$  is initialized as a copy of  $S$ );
6        $Berl \leftarrow BM(s, L, j)$ ;
7       ( $BM$  is a subroutine which returns the shortest recurrence
8       relation on  $L$  terms of  $s$ , starting at  $j$ )
9        $i \leftarrow 1$ ;
10       $e \leftarrow [0, \dots, 0]$ ;
11      while  $K \leq k$  and  $i \leq n$  do
12         $d \leftarrow \sum_{i'=0}^{Berl[1]} Berl[2][1 + i'] * s[(i + i') \pmod n]$ ;
13        if  $d \leftarrow 0$  then
14           $e[(i + Berl[1]) \pmod n] \leftarrow 0$ ;
15           $i \leftarrow i + 1$ ;
16        else
17           $s[(i + Berl[1]) \pmod n] \leftarrow$ 
18             $s[(i + Berl[1]) \pmod n] - d$ ;
19           $e[(i + Berl[1]) \pmod n] \leftarrow -d$ ;
20           $i \leftarrow i + 1$ ;
21           $K \leftarrow K + 1$ ;
22        end
23      end
24      if  $K \leq k$  then
25         $Result[Length(Result) + 1] \leftarrow Berl[1]$ ;
26      end
27    end
28  end
29  Return(Result);
30 end

```

$k =$	0	1	2	3
Average Linear Complexity:	15.04	12.90	11.14	9.42
Number of Sequences Returning Correct Linear Complexity:	50	31	22	18
Average Difference Between Returned and Optimal Linear Complexity:	-	0.56	0.88	1.14
Difference as a Fraction of the Linear Complexity:	-	0.04	0.08	0.12
Average Difference for Sequences that do not Return the optimal Linear Complexity:	-	1.47	1.57	1.78
Difference as a fraction of the Linear Complexity:	-	0.12	0.14	0.19

Table 4.5: Numerically Generated Results, on 50 sequences, each of length 30

Lemma 4.10. *Let f_k be defined as follows:*

$$f_k(N, n) = \begin{cases} 1 & k = 0 \\ N(q-1) & k = 1 \\ \sum_{i=2n}^{N-k} N \binom{N-2-i}{k-2} (q-1)^k & k \geq 2 \end{cases}$$

Then when $N/2 \leq 2n < N$, the number of error sequences with gap of length $2n$ or with at least $2n$ leading and trailing zeros is given by

$$\sum_{i=0}^k f_i(N, n)$$

Proof. This lemma can be seen as the analogue of Lemma 4.3. Again the three cases will be considered separately.

When $k = 0$, as before, there is only one error sequence, and the algorithm will find the optimal recurrence.

When $k = 1$, wherever the error is located in the sequence, there must be a gap of size $N - 1$, that is, all the other terms in the sequence. Therefore, provided $2n < N$, $kErApFP$ will find the optimal error sequence, which is equivalent to searching through all N possible locations for the error, and all $(q - 1)$ non-zero values for it.

When $k \geq 2$ all error sequences will have their largest gap between two non-zero elements. This gap can begin at any point in the sequence, giving N different possibilities. We sum over the different lengths of the gap, and note that once the

N=32, k = 1				
	n=8	10	12	14
Total Space Size	33	33	33	33
Searched Space	33	33	33	33
Percentage	100.0%	100.0%	100.0%	100.0%
N=32, k = 2				
	n=8	10	12	14
Total Space Size	529	529	529	529
Searched Space	513	385	257	129
Percentage	97.0%	72.8%	48.6%	24.4%
N=32, k = 3				
	n=8	10	12	14
Total Space Size	5489	5489	5489	5489
Searched Space	3873	2145	929	225
Percentage	70.6%	39.1%	16.9%	4.1%

Table 4.6: Size of the search Space, N=32

size of the gap has been fixed, and two of the non-zero elements have been fixed at either end of the gap, we are left to place the remaining $k - 2$ errors freely in the remaining $N - i - 2$ positions, where i is the size of the gap. As before, for each possible error there are $(q - 1)$ possible, non-zero values for it. \square

We also note that since the search space for $kErApFP$ is at least as big as the search space for $kErAp$ that it also searches an exponentially large space, and that the running time for the algorithm is unchanged, that is, $\mathcal{O}(N^4)$.

4.7 Numerical Results

In Tables 4.6, 4.7 and 4.8 we tabulate the size of the space searched by $kErApFP$ for various values of N , n and k . As before only sequences over $\text{GF}(2)$ are considered.

From these tables, and the lemmas above, it can be seen that there is a dramatic increase in the number of error sequences that are considered by $kErApFP$ when compared to $kErAp$. However, there is the trade-off that it is necessary to assume that the sequence that is used as the input for the algorithm is the whole period of the sequence. Whether or not this is suitable will be dependent on the specific instance that the algorithm is being used in.

As before, this algorithm does not provide any assurances about the closeness of the result when the optimal relation is not found. Therefore we have run the algorithm on 50 randomly generated sequences, each of length 30, and the results are shown in Table 4.9. The first thing to note about these results is that there is a dramatic increase in the average linear complexity of the sequences. This is caused

N=48, k = 1						
	n=12	14	16	18	20	22
Total Space Size	49	49	49	49	49	49
Searched Space	49	49	49	49	49	49
Percentage	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
N=48, k = 2						
	n=12	14	16	18	20	22
Total Space Size	1177	1177	1177	1177	1177	1177
Searched Space	1153	961	769	577	385	193
Percentage	98.0%	81.6%	65.3%	49.0%	32.7%	16.4%
N=48, k = 3						
	n=12	14	16	18	20	22
Total Space Size	18770	18770	18770	18770	18770	18770
Searched Space	13297	9169	5809	3217	1393	337
Percentage	72.0%	49.6%	31.4%	17.4%	7.5%	1.8%

Table 4.7: Size of the search Space, N=48

N=60, k = 1							
	n=16	18	20	22	24	26	28
Total Space Size	61	61	61	61	61	61	61
Searched Space	61	61	61	61	61	61	61
Percentage	100%	100%	100%	100%	100%	100%	100%
N=60, k = 2							
	n=16	18	20	22	24	26	28
Total Space Size	1831	1831	1831	1831	1831	1831	1831
Searched Space	1681	1441	1201	961	721	481	241
Percentage	91.8%	78.7%	65.6%	52.5%	39.4%	26.3%	13.2%
N=60, k = 3							
	n=16	18	20	22	24	26	28
Total Space Size	36051	36051	36051	36051	36051	36051	36051
Searched Space	22741	16621	11461	7261	4021	1741	421
Percentage	63.1%	46.1%	31.8%	20.1%	11.2%	4.8%	1.2%

Table 4.8: Size of the Search Space, N=60

$k =$	0	1	2	3
Average Linear Complexity:	27.09	21.78	17.92	14.60
Number of Sequences Returning Correct Linear Complexity:	50	6	3	5
Average Difference Between Predicted and Actual Linear Complexity:	-	4.40	5.80	6.78
Difference as a Fraction of the Linear Complexity:	-	0.22	0.34	0.47
Average Difference for Sequences that do not Return the optimal linear complexity:	-	5.00	6.17	7.53
Difference as a fraction of the linear complexity:	-	0.24	0.36	0.53

Table 4.9: Numerically Generated Results, on 50 sequences, each of length 30

by the relation needing to generate the entire sequence, and looping round to the beginning again. As we have seen before, an increase in the linear complexity of the relation will result in a smaller number of error sequences being considered by the algorithm, and so this explains the higher values for the differences in the predicted and optimal linear complexity.

4.8 Analysis of Results

We have seen that the performance of the algorithm varies depending on the situation in which it is being applied and the particular sequence upon which it is running. Specifically, while we have enumerated the set of sequences on which it is guaranteed to find the optimal error sequence, there are many sequences for which it does not, and on these sequences, we can provide no guarantees about the closeness of the approximation. Whilst it does appear to give a close approximation for small sequences, the performance is less good for longer sequences, and also when considering more errors. We have also looked at the size of the search space and seen that as the sequence length increases, or the linear complexity increases, the performance of the algorithm worsens. Whilst in some situations the entire set of error sequences was considered, in the worst case we studied (finite sequence, $N = 60$, $k = 3$, $n = 28$) the performance was poor. Finally, we can see that whilst the algorithm searched a smaller set of error sequences in the finite case, its performance in terms of approximations was less good on average, when compared to the periodic case.

Chapter 5

Linear Complexity for Sequences with Characteristic Polynomial of the Form f^v for an Irreducible f

As discussed in Chapter 3 the Berlekamp-Massey algorithm computes the linear complexity and minimal polynomial of a sequence in quadratic time, and for certain classes of sequences more efficient algorithms exist. The Games-Chan algorithm [28] takes linear time and works for binary sequences with period of the form 2^n . It exploits the fact that in this case the minimal polynomial is a factor of $x^{2^n} - 1 = (x - 1)^{2^n}$; hence it is a power of $x - 1$ and it is only needed to determine which power. The Games-Chan algorithm assumes that a whole (not necessarily minimal) period of the sequence is known. In Algorithm 3 we generalise the Games-Chan algorithm to the case when it is known a priori that the minimal polynomial is a power of a certain fixed irreducible polynomial f (so the Games-Chan algorithm would be the case $f = x - 1$).

The Games-Chan algorithm has been generalised to fields of arbitrary characteristic by Ding *et al.* in [22] and we will similarly give Algorithm 4 which is the generalisation of Algorithm 3 to arbitrary characteristic. (Our algorithm reduces to the one of Ding *et al.* in [22] when $f = x - 1$.)

It was noted by Sălăgean in [64] and by Meidl in [54] that it is actually not necessary to have a whole period of the sequence in order to determine its linear complexity using the Games-Chan algorithm. It suffices to have a number of terms greater or equal to the linear complexity, provided we still know that the sequence admits as a characteristic polynomial a power of $x - 1$ or more generally of some irreducible polynomial f . For finite sequences which have a characteristic polynomial of the form f^v Meidl gives two algorithms in [54]: one for $f = x - 1$ and arbitrary v , the other for arbitrary f and v being a power of 2. We generalise

his approach as Algorithm 6, which works for arbitrary f and arbitrary v . At first sight it would seem tempting to take this generalisation further, to k -error linear complexity, as in [54, Section 4]. However, we do not feel that such work would be worthwhile, as the definition used for the k -error linear complexity in [54] is a restricted one (it computes the minimum linear complexity of all sequences z at Hamming distance k from s with the additional condition that z admits as a characteristic polynomial a power of f), and is not equivalent to the generally used definition (except for $f = x - 1$). A further explanation is contained in Remark 5.13.

All the algorithms mentioned so far have linear computational complexity, like the Games-Chan algorithm.

We further generalise our algorithms for infinite sequences to determine the minimal polynomial when all its irreducible factors are known a priori (Algorithm 7). This algorithm is efficient only if the number of irreducible factors is small.¹

5.1 Linear Complexity of Infinite Sequences with a Characteristic Polynomial a Power of an Irreducible Polynomial

We assume that we are given an infinite sequence s of (not necessarily minimal) period N over a field K of characteristic p and that we know that the sequence admits as a characteristic polynomial a power of a fixed irreducible polynomial f , i.e. $s \in \mathcal{M}(f^\infty)$. Our goal is to determine the minimal polynomial of s , which will obviously be of the form f^r for some integer r . A naive algorithm could compute $f^i s$ for increasing values of i (by repeatedly replacing s by fs) until the zero sequence is obtained. The more efficient method described here finds an upper bound for r and then does a p -ary search for the value of r .

An upper bound on the value of r can be obtained by looking at the period N and using Theorem 2.16:

Lemma 5.1. *Let $s \in \mathcal{M}(f^\infty)$, $s \neq \mathbf{0}$ and let N be a (not necessarily minimal) period of s . Write N as $N = p^w N'$ with $p \nmid N'$. Then $s \in \mathcal{M}(f^{p^w})$, $\text{ord}(f) \mid N'$ and $p^w \text{ord}(f)$ is also a period of s .*

Proof. Let f^r be the minimal polynomial of s and let w' be the smallest integer with $p^{w'} \geq r$. The minimal period of s is $\text{ord}(f^r)$, which by Theorem 2.16 equals $p^{w'} \text{ord}(f)$. Any other period of s , for example $N = p^w N'$ is a multiple thereof.

¹The main results of this Chapter were originally published in [19].

Since neither $\text{ord}(f)$ nor N' are divisible by p , this means $w' \leq w$ and $\text{ord}(f) \mid N'$. Hence $r \leq p^{w'} \leq p^w$ so $s \in \mathcal{M}(f^{p^w})$ and $p^w \text{ord}(f)$ is a period of s . \square

Once we have an upper bound for r we can find the exact value of r by a p -ary search. We actually find the largest exponent i for which $f^i s \neq \mathbf{0}$, i.e. $r - 1$. To obtain r , a correction of $+1$ is added at the end. One can view the p -ary search equivalently as determining the digits of the base p representation of $r - 1$. When testing whether $f^i s \neq \mathbf{0}$ for different values of i , the values of i which are powers of p are preferred for efficiency reasons, as they minimise the weight of f^i , as is shown in Lemma 2.11.

Lemma 5.2. *Let $s \in \mathcal{M}(f^\infty)$, $s \neq \mathbf{0}$ and let $N = p^w N'$ with $p \nmid N'$ be a (not necessarily minimal) period of s (which by Lemma 5.1 means $s \in \mathcal{M}(f^{p^w})$). Let f^r be the minimal polynomial of s . If $w = 0$ then $r = 1$. For $w \geq 1$, let $r - 1 = r_{w-1}p^{w-1} + r_{w-2}p^{w-2} + \dots + r_1p + r_0$ with $r_i \in \{0, 1, \dots, p-1\}$ be the representation of $r - 1$ in base p . Then we have: r_{w-1} is the largest integer $i \geq 0$ for which $f^{ip^{w-1}} s \neq \mathbf{0}$.*

Moreover, putting $t = f^{r_{w-1}p^{w-1}} s$ we have $t \in \mathcal{M}(f^{p^{w-1}})$, t has minimal polynomial $f^{r-r_{w-1}p^{w-1}}$ and period N/p .

Proof. It is easy to see that $r_{w-1}p^{w-1} \leq r - 1 < (r_{w-1} + 1)p^{w-1}$, hence $r_{w-1}p^{w-1} < r \leq (r_{w-1} + 1)p^{w-1}$. So $f^{r_{w-1}p^{w-1}}$ is not a characteristic polynomial of s , whereas $f^{(r_{w-1}+1)p^{w-1}}$ is. By Lemma 2.10(i), this means $f^{r_{w-1}p^{w-1}} s \neq \mathbf{0}$ and $f^{(r_{w-1}+1)p^{w-1}} s = \mathbf{0}$.

The last equality also means that $f^{p^{w-1}} t = \mathbf{0}$, i.e. $t \in \mathcal{M}(f^{p^{w-1}})$. By Theorem 2.16, $p^{w-1} \text{ord}(f)$ is a period of t . By Lemma 5.1, we know $\text{ord}(f) \mid N'$. Hence $p^{w-1} N' = N/p$ is also a period of t . \square

From Lemmas 5.1 and 5.2 we have:

Corollary 5.3. *With the notations of Lemma 5.2: $r_{w-1} = 0$ iff $f^{p^{w-1}} s = \mathbf{0}$ iff $s \in \mathcal{M}(f^{p^{w-1}})$ iff s has period N/p .*

Based on the Lemmas and Corollary above we present the following two algorithms LinCompChar2 and LinComp given as Algorithms 3 and 4. The first is for $p = 2$ and the second for arbitrary p (including $p = 2$). We formulated the algorithm for $p = 2$ separately due to the importance of characteristic 2 and similarity with Games-Chan algorithm, as well as the fact that the code is somewhat simplified by the existence of only 2 cases at each step, corresponding to a 0/1 digit in the binary representation of $r - 1$, so the test in Corollary 5.3 is sufficient for determining the digit.

Note that throughout the algorithms the current value of the infinite sequence s is implicitly stored as being the finite sequence $s' = (s_0, s_2, \dots, s_{N-1})$ of length N repeated periodically. When computing the action of a polynomial say $g = \sum_{i=0}^n a_i x^i$ on the infinite s thus stored, the result will be an infinite sequence t of period N stored as the finite sequence $t' = (t_0, t_2, \dots, t_{N-1})$ consisting of the first N terms of t , computed from s' as $t_i = \sum_{j=0}^n a_j s_{(i+j) \bmod N}$.

Algorithm 3: LinCompChar2(s', N, f)

Input : $f \in K[x]$ an irreducible polynomial over a field K of characteristic 2; $s' = (s_0, \dots, s_{N-1})$ a finite sequence over K consisting of the first N terms of an infinite sequence s of (not necessarily minimal) period N such that $s \in \mathcal{M}(f^\infty)$.

Output: The minimal polynomial of the sequence s

```

1 begin
2    $C = 0$ ;
3   if  $s' = \mathbf{0}$  then
4     | return( $f^C$ );
5   end
6    $w =$  the largest integer for which  $2^w$  divides  $N$ ;
7   Optionally, if  $\text{ord}(f)$  precomputed, set  $N = 2^w \text{ord}(f)$  and
    $s' = (s_0, s_1, \dots, s_{N-1})$ ;
8   while  $w \geq 1$  do
9     |  $t' = f^{2^{w-1}} s'$  (as action on an infinite sequence);
10    | if  $t' \neq \mathbf{0}$  then
11      |  $s' = t'$ ;
12      |  $C = C + 2^{w-1}$ ;
13    | end
14    |  $s' = (s_0, s_1, \dots, s_{N/2-1})$ ;
15    |  $w = w - 1$ ;
16    |  $N = N/2$ ;
17  end
18   $C = C + 1$ ;
19  return( $f^C$ );
20 end
```

Theorem 5.4. *Algorithms LinCompChar2 and LinComp (Algorithms 3 and 4) are correct and terminate.*

Proof. Let r be such that f^r is the minimal polynomial of the input sequence, i.e. the expected output of the algorithm. For the correctness of both algorithms we can prove the following invariants. At the start and at the end of each run of the outer while loop, s' is a finite sequence of length N , and $s' \neq \mathbf{0}$. If we denote by s the infinite sequence of period N obtained by repeating s' , we have $s \in \mathcal{M}(f^{p^w})$ and the minimal polynomial of s is f^{r-C} .

Algorithm 4: LinComp(s', N, f)

Input : $f \in K[x]$ an irreducible polynomial over a field K of characteristic p ; $s' = (s_0, \dots, s_{N-1})$ a finite sequence over K consisting of the first N terms of an infinite sequence s of (not necessarily minimal) period N such that $s \in \mathcal{M}(f^\infty)$.

Output: The minimal polynomial of the sequence s

```

1 begin
2    $C = 0$ ;
3   if  $s' = \mathbf{0}$  then
4     | return( $f^C$ );
5   end
6    $w =$  the largest integer for which  $p^w$  divides  $N$ ;
7   Optionally, if  $\text{ord}(f)$  precomputed, set  $N = p^w \text{ord}(f)$  and
    $s' = (s_0, s_1, \dots, s_{N-1})$ ;
8   while  $w \geq 1$  do
9     |  $t' = f^{p^{w-1}} s'$  (as action on an infinite sequence);
10    | while  $t' \neq \mathbf{0}$  do
11      | |  $s' = t'$ ;
12      | |  $C = C + p^{w-1}$ ;
13      | |  $t' = f^{p^{w-1}} s'$  (as action on an infinite sequence);
14    | end
15    |  $s' = (s_0, s_1, \dots, s_{N/p-1})$ ;
16    |  $w = w - 1$ ;
17    |  $N = N/p$ ;
18  end
19   $C = C + 1$ ;
20  return( $f^C$ );
21 end
```

We outline the proofs for these invariants. The fact that s' is a finite sequence of length N is immediate. The fact that $s' \neq \mathbf{0}$ is obviously true before the while loop, due to the first if statement, line 3. If s' is non-zero at the start of the outer while loop it will stay non-zero throughout, as each new value of s' is always set to either the first N/p elements of s' (and by Corollary 5.3, s' consists in this case of p repeating identical sequences, which are therefore non-zero) or to a non-zero value of t' .

The fact that $s \in \mathcal{M}(f^{p^w})$ and the minimal polynomial of s is f^{r-C} holds before the outer while loop due to Lemma 5.1 and is then maintained due to Lemma 5.2.

Finally, upon exiting the outer while loop, we have that $w = 0$ so $s \in \mathcal{M}(f)$. Since $s' \neq \mathbf{0}$, this means f is the minimal polynomial of s . On the other hand we saw that the minimal polynomial of s is f^{r-C} , therefore $1 = r - C$ at this point, so $C + 1$ is correctly returned as r .

The termination follows from the fact that the value of w is decreased by one at each run of the while loop. Additionally, for characteristic p , the inner while loop will run at most $p - 1$ times, as we know that at the beginning of each run of the outer while loop we have $s \in \mathcal{M}(f^{p^w})$ hence $f^{p^w} s = \mathbf{0}$.

□

Theorem 5.5. *The complexity of Algorithms LinCompChar2 and LinComp (Algorithms 3 and 4) is $\mathcal{O}(N)$ if we consider f to be fixed, or $\mathcal{O}(\text{wt}(f)N)$ if f is an input parameter.*

Proof. It suffices to show the second result. Let N_0 be the initial value of N , w_0 be the initial value computed for w and let $N' = N_0/p^{w_0}$. The (outer) while loop will run w_0 times, as showed in the proof of Theorem 5.4.

In the binary case, the complexity of each individual loop is dominated by the calculation of $t' = f^{2^{w-1}} s'$, a finite sequence representing the first $2^w N'$ terms of an infinite sequence of period $2^w N'$. The number of summands for each term is fixed by Lemma 2.11 as $\text{wt}(f)$. So the number of arithmetic operations is $2^w N'(\text{wt}(f) - 1)$. For characteristic p each run of the outer while loop consists of at most p computations of $t' = f^{p^{w-1}} s'$, each taking $(\text{wt}(f) - 1)p^w N'$ steps.

In total we have $\sum_{w=1}^{w_0} 2^w N'(\text{wt}(f) - 1) = 2(2^{w_0} - 1)N'(\text{wt}(f) - 1) < 2N_0 \text{wt}(f)$ for the binary case and $(\text{wt}(f) - 1)N' \sum_{w=1}^{w_0} p^{w+1} = (\text{wt}(f) - 1)p^2 N' \frac{p^{w_0} - 1}{p-1} < \frac{p^2}{p-1} \text{wt}(f)N_0$ for arbitrary p .

□

Alternative algorithms can be obtained for LinCompChar2 and LinComp (Algorithms 3 and 4) by using the last equivalence of Corollary 5.3. Namely, we can check immediately at the start of the outer while loop whether the current value of s' consists of p repeating copies of the same sequence. If this is the case we do

not compute t' but skip to the instructions for updating the values of s', w and N at the end of the loop. The algorithms thus modified would have the same worst-case complexity but will behave slightly better for the case when $r - 1$ has many 0's in its representation in base p . Since each digit has a $1/p$ chance of being 0, the savings will be more significant when the underlying field has characteristic 2. We present an alternative to Algorithm 3 as Algorithm 5.

Algorithm 5: LinCompAlternative(s', N, f)

Comment: Same as algorithm LinCompChar2 except that lines 8-17 are replaced by

```

8 while  $w \geq 1$  do
9   if  $(s_0, s_1, \dots, s_{N/2-1}) \neq (s_{N/2}, s_{N/2+1}, \dots, s_{N-1})$  then
10    |  $s' = f^{2^{w-1}} s'$  (as action on an infinite sequence);
11    |  $C = C + 2^{w-1}$ ;
12  end
13   $s' = (s_0, s_1, \dots, s_{N/2-1})$ ;
14   $w = w - 1$ ;
15   $N = N/2$ ;
16 end

```

Remark 5.6. For $f = x - 1$, Algorithm 3 reduces to the Games-Chan algorithm, [28]. Firstly, $\text{ord}(f) = 1$, so we can put $N = 2^w$ in line 7. Secondly, computing $t' = (x - 1)^{2^{w-1}} s' = (x^{2^{w-1}} - 1)s'$ for a sequence of period 2^w means t' is the component-wise subtraction of the two halves of s' (i.e. t' is $L(s) - R(s)$ if $L(s)$ and $R(s)$ denote the left and right half of s , as in the notation used in the Games-Chan algorithm). Therefore checking whether $t' = \mathbf{0}$ will in this case mean checking whether the two halves of s' are identical. Algorithm 5 will also become virtually the same as Algorithm 3 and the same as Games-Chan algorithm in this situation. Similarly, for $f = x - 1$ Algorithm 4 reduces to the algorithm of Ding et al. [22].

Note that in the Games-Chan algorithm the final instruction $C = C + 1$ is done conditionally, only if $s' \neq \mathbf{0}$. If one deals at the start of the algorithm with the case of an all-zero input sequence (as we do), it is no longer necessary to check at the end if $s' \neq \mathbf{0}$, as this will always be the case (see the proof of Theorem 5.4).

Example 5.7. Let $K = \text{GF}(2)$ and $f = x^3 + x + 1$. The sequence $s \in \mathcal{M}(f^\infty)$ has period $N = 28$ and its first 28 terms are $s' = 0000000 0101100 0010111 0111011$. The running of Algorithm 3 is described in Table 5.1.

Table 5.1: Example Running of LinComp

s'	w	$t = \mathbf{0}$?	C
0000000 0101100	2	No	2
0010111 0111011			
0010111 0010111	1	Yes	2
0010111	0		
$C = C + 1$			3
return(f^3)			

5.2 Linear Complexity of Finite Sequences with Characteristic Polynomial a Power of an Irreducible Polynomial

It was noticed by Sălăgean in [64] and by Meidl in [54] that we actually do not need to have the whole period of the infinite sequence in the Games-Chan algorithm in order to compute the linear complexity. In this section we generalise the idea of Meidl, [54, Sections 2 and 3], by using the initial terms of a sequence, and knowledge of a characteristic polynomial to treat it as an infinite sequence.

For a fixed polynomial g , an individual infinite sequence s with characteristic polynomial g is uniquely defined (within the class of all sequences with characteristic polynomial g) by its initial $\deg(g)$ terms. Can we decide if s admits a characteristic polynomial of lower degree just by examining these initial $\deg(g)$ terms?

Lemma 5.8. *Let s be an infinite sequence with characteristic polynomial $g = g_1g_2$ with g_1, g_2 monic. Then s has characteristic polynomial g_1 iff $s' = (s_0, \dots, s_{\deg(g)-1})$ has characteristic polynomial g_1 .*

Proof. The direct implication is obvious. Conversely, assume s' has characteristic polynomial g_1 i.e. $g_1s' = (0, \dots, 0)$, a finite sequence of $\deg(g) - \deg(g_1) = \deg(g_2)$ terms. Note this sequence also coincides with the first $\deg(g_2)$ terms of the infinite sequence g_1s . By Lemma 2.10(i), $g_1s = g_2g_1s = \mathbf{0}$, so g_1s has characteristic polynomial g_2 . But then $g_1s = \mathbf{0}$ as its first $\deg(g_2)$ terms are all zero and its linear complexity is at most $\deg(g_2)$. □

Consequently, if we are given s' as being the first $v \deg(f)$ terms of a sequence $s \in \mathcal{M}(f^v)$ we can check whether s admits some characteristic polynomial of lower degree, i.e. $f^{v'}$ with $v' < v$ by checking whether $f^{v'}s' = \mathbf{0}$. Again, a p -ary search will make it more efficient.

The algorithm LinCompFinite is given as Algorithm 6 and is similar to the Algorithm 4 in the previous section. Note that throughout the algorithm, the

length of the current value of s' is $v \deg(f)$ for the current value of v .

Algorithm 6: $\text{LinCompFinite}(s', v, f)$

Input : A finite sequence s' consisting of the first $v \deg(f)$ elements of an infinite sequence $s \in \mathcal{M}(f^v)$ where $f \in K[x]$ is a fixed irreducible polynomial over a field K of characteristic p .

Output: The minimal polynomial of the sequence s

```

1 begin
2    $C = 0$ ;
3   if  $s' = \mathbf{0}$  then
4     | return( $f^C$ );
5   end
6    $w =$  the smallest integer such that  $v \leq p^w$ ;
7   while  $w \geq 1$  do
8     |  $t' = f^{p^{w-1}}s'$  (as action on a finite sequence);
9     | if  $t' \neq \mathbf{0}$  then
10    |   |  $s' = t'$ ;
11    |   |  $C = C + p^{w-1}$ ;
12    |   |  $v = v - p^{w-1}$ ;
13    |   |  $w =$  the smallest integer such that  $v \leq p^w$ ;
14    |   else
15    |   |  $v = p^{w-1}$ ;
16    |   |  $w = w - 1$ ;
17    |   |  $s' = (s_0, s_1, \dots, s_{v \deg(f)-1})$ ;
18    |   end
19  end
20   $C = C + 1$ ;
21  return( $f^C$ );
22 end
```

Theorem 5.9. *Algorithm LinCompFinite is correct and terminates.*

Proof. Let $s^{(0)}$ be the original value of the infinite input sequence s and let r be such that f^r is its minimal polynomial. For the correctness, we will show that throughout the algorithm s' consists of the first $v \deg(f)$ terms of $f^C s^{(0)}$ and $p^w \geq v \geq r - C \geq 1$, with w minimal such that $p^w \geq v$. Therefore s' in conjunction with the characteristic polynomial f^v correctly defines the infinite sequence $f^C s^{(0)}$. All these statements are obviously true before the while loop begins. We will assume they are true at the start of a run on the loop and show they are true at the end of the loop.

If the $t' \neq \mathbf{0}$ branch of the **if** is taken, then $v_{new} = v_{old} - p^{w_{old}-1} \geq r - C_{old} - p^{w_{old}-1} = r - C_{new}$, where the *old* and *new* indices refer to the value at the beginning and at the end of the loop. In addition, $r - C_{new} \geq 1$: if we assume the contrary, i.e. $r \leq C_{new}$, then $f^{C_{new}}$ would be a characteristic polynomial for $s^{(0)}$,

Table 5.2: Example running of LinCompFinite

s'	w	v	$t = \mathbf{0}?$	C
010100001011010110	3	6	Yes	0
010100001011	2	4	No	2
001111	1	2	No	3
101	0	1		
$C = C + 1$				4
return(f^4)				

so $f^{C_{new}}s^{(0)} = \mathbf{0}$. But then s'_{new} , consisting of the first terms of $f^{C_{new}}s^{(0)}$, would equal $\mathbf{0}$. On the other hand $s'_{new} = t' \neq \mathbf{0}$, giving a contradiction.

If the else branch of the **if** is taken, we know $f^{p^{w_{old}-1}}s'_{old} = \mathbf{0}$. By Lemma 5.8, this means $f^{C_{old}}s^{(0)}$ has characteristic polynomial $f^{p^{w_{old}-1}} = f^{v_{new}}$, so $v_{new} \deg(f)$ of its initial terms are sufficient to determine the sequence. Moreover, since $f^{r-C_{old}}$ is the minimal polynomial of $f_{old}^C s^{(0)}$ we have $p^{w_{old}-1} \geq r - C_{old}$. Hence $v_{new} = p^{w_{old}-1} \geq r - C_{old} = r - C_{new}$.

Finally, upon exiting the while loop we have $w = 0$ and since $1 = p^w \geq v \geq r - C \geq 1$, it follows that $r - C = 1$, so $C + 1$ is correctly returned as r .

To show termination, it suffices to note that v decreases throughout the algorithm. □

Theorem 5.10. *Algorithm LinCompFinite has complexity $\mathcal{O}(v)$ for a fixed f , or $\mathcal{O}(v \deg(f) \text{wt}(f))$ if f is an input parameter.*

Proof. We can see from the proof of Theorem 5.9 that each run of the while loop takes $v \deg(f) \text{wt}(f)$ steps for the current value of v , which is upper bounded by p^w . In each run of the loop, the value of w is decreased or stays the same, but can only stay the same for at most p successive runs of the while loop. So if w_0 is the initial value of w , we have at most $\sum_{w=1}^{w_0} p^{w+1} \deg(f) \text{wt}(f)$ steps. □

Example 5.11. *Let $K = GF(2)$ and $f = x^3 + x + 1$. The finite sequence $s' = 010100001011010110$ consists of the first $6 \deg(f) = 18$ terms of a sequence $s \in \mathcal{M}(f^6)$. The running of Algorithm 6 is described in the Table 5.2.*

Remark 5.12. *For $f = x - 1$, $p = 2$ and arbitrary v , our Algorithm 6 reduces to Algorithm 1 of [54]. For $f = x^2 + x + 1$ and v being a power of 2, it reduces to Algorithm 2 in [54] (which, as was remarked at the end of Section 3 of [54] could be generalised to arbitrary f and v a power of 2).*

Let us examine the relation between the algorithms in this section (where a finite portion of the sequence is known) and the ones in the previous section (where a whole period of the sequence is known, i.e. the whole infinite sequence is known). We could easily transform one problem into the other, namely, if we have a finite

sequence we can generate the whole period using the given characteristic polynomial f^v (note that this process can take a number of steps exponential in the length of the original input finite sequence, see discussion further on). Conversely given an infinite sequence of period N we could restrict to the initial $p^w \deg(f)$ terms (with w maximal such that $p^w | N$), as f^{p^w} is guaranteed by Lemma 5.1 to be a characteristic polynomial of f . We compare now the complexities of the two types of algorithm. Algorithms 3, 4 and 5 of the previous section are $\mathcal{O}(N \text{wt}(f))$, so since $p^w \text{ord}(f) | N$, they are $\mathcal{O}(p^w \text{ord}(f) \text{wt}(f))$. On the other hand, Algorithm 6 of this section has complexity $\mathcal{O}(p^w \deg(f) \text{wt}(f))$. Since $\deg(f) \leq \text{ord}(f) \leq p^{\deg(f)} - 1$ with both lower and upper bounds attained for particular values of f , it means that the algorithms of the previous section are no better, and potentially exponentially slower than the ones in this section (to clarify, all are linear in the size of the input, but the size of the input can be exponentially higher if we use the full period rather than the initial $v \deg(f)$ terms). So the algorithms of the previous section should be avoided in favour of the one in this section. We did nevertheless present them as they are direct generalisations of the Games-Chan algorithm.

Remark 5.13. *Since the algorithm developed here is a generalization of the ones given by Meidl in [54], it may seem natural to go further and adapt these algorithms into ones capable of determining the k -error linear complexity, as a generalization of the work carried out in [54], Section 4. However, we do not believe that such work would be worthwhile for the following reason:*

In [64] the following definition is given for the k -error linear complexity of a finite sequence, z , of length t with respect to a set A of infinite sequences:

$$\text{LC}_k(z, A) = \min\{\text{LC}(s) \mid s \in A, \text{wt}((s_0, s_1, \dots, s_{n-1}) - z) \leq k\}$$

where $\text{LC}(s)$ is the linear complexity of the infinite sequence s . This definition is used in [64] with the set A being the set of all sequences with period a power of 2. This is because, if it is known that the initial finite sequence was part of an infinite sequence s whose period was a power of 2, introducing errors to this infinite sequence would not affect this property (because the error sequence would have the same period as s , i.e. a power of two and hence adding them to the sequence would again result in a sequence with period a power of two), and so $\text{LC}_k(z, A) \equiv \text{LC}_k(z, \mathcal{S})$ where \mathcal{S} is the set of all binary sequences. However, this is not always the case, and specifically, it is not the case in the way the definition is used in [54, Section 4].

In [54, Section 4], the same general definition is used, although now the set A is defined to be $\mathcal{M}((x^2 + x + 1)^{2^v})$. However, $s \in \mathcal{M}((x^2 + x + 1)^{2^v})$ does not imply $(s + e) \in \mathcal{M}((x^2 + x + 1)^{2^v})$, for all sequences e of the same period as s . Hence

in this case $c_k(s, \mathcal{M}((x^2 + x + 1)^{2^v})) \neq \text{LC}_k(s, \mathcal{S})$. Therefore while the algorithm presented in [54, Section 4] does calculate $\text{LC}_k(s, \mathcal{M}((x^2 + x + 1)^{2^v}))$, this is not equal to the k -error linear complexity (in the classical sense) of the input sequence s . Therefore we do not feel it is worthwhile adapting the algorithm presented in this paper to the k -error linear complexity problem, as it would suffer from the same restriction.

5.3 Linear Complexity of Infinite Sequences with a Characteristic Polynomial a Product of Known Irreducible Factors

With a simple adjustment to the algorithms in Section 5.1, we can greatly increase their scope, so that they can be applied to any sequence provided each of the irreducible factors of a characteristic polynomial are known.

As a consequence of Lemma 2.10(ii) we have:

Corollary 5.14. *Assume that the minimal polynomial of a sequence s is of the form $f_1^{r_1} f_2^{r_2} \dots f_m^{r_m}$, with f_i distinct irreducible polynomials. Let $x_i \geq r_i$ for $i = 2, \dots, m$. Then $f_1^{x_1}$ is a minimal polynomial of the sequence $f_2^{x_2} \dots f_m^{x_m} s$.*

Therefore, if we know each of the irreducible polynomials which divides a characteristic polynomial of a sequence and we have an upper bound on the powers of each irreducible polynomial, we can use Corollary 5.14 and Algorithms 3 or 4, to successively determine the powers of each irreducible polynomial in the minimal polynomial. To obtain an upper bound of the power of each irreducible polynomial, note that the minimal polynomial is a factor of $x^N - 1$ where N is a period of s . Writing $N = p^w N'$ with $p \nmid N'$ we have $x^N - 1 = (x^{N'} - 1)^{p^w}$. Putting $\Phi_{N'} = \{f \in K[x] \mid f \text{ irreducible factor of } x^{N'} - 1\}$ all irreducible factors of the minimal polynomial are in $\Phi_{N'}$ and have multiplicity at most p^w . The resulting algorithm LinCompSet is presented as Algorithm 7.

Theorem 5.15. *For a sequence of period N , and a fixed set Φ , Algorithm 7 has complexity $\mathcal{O}(N)$. For a general set of m elements $\Phi = \{f_1, \dots, f_m\}$, the algorithm has complexity $\mathcal{O}((\sum_{i=1}^m \text{wt}(f_i))mN)$.*

Proof. In each of the m runs of the outer for loop, the computation of t' takes $((\sum_{j=1}^m \text{wt}(f_j)) - \text{wt}(f_i))N$. By Theorem 5.5, LinComp has complexity $\mathcal{O}(\text{wt}(f_i)N)$ so a total of $\mathcal{O}(N \sum_{i=1}^m \text{wt}(f_i))$ for each loop. \square

Note that Algorithm 7 is therefore efficient only if Φ has a small cardinality and the total weight of its elements is small. We could remove the condition that

Algorithm 7: $\text{LinCompSet}(s', N, \Phi = \{f_1, \dots, f_m\})$

Input : s' a finite sequence consisting of the first N terms of an infinite sequence s of (not necessarily minimal) period N ; Φ a superset of the set of all irreducible factors of the minimal polynomial of s .

Output: The minimal polynomial of s

```

1 begin
2    $w =$  the largest integer for which  $p^w$  divides  $N$ ;
3    $g = 1$ ;
4   for  $i = 1, 2, \dots, m$  do
5     for  $j = 1, 2, \dots, m$  do
6       if  $j \neq i$  then
7          $t' = f_j^{p^w} s'$  (as action on infinite sequences);
8       end
9     end
10     $g = g * \text{LinComp}(t', N, f_i)$ ;
11  end
12  return( $g$ );
13 end
```

Φ must be known in advance by computing $\Phi_{N'}$ as above and using it as Φ during the algorithm. Hence we would have $m = |\Phi_{N'}|$. The cases of interest will be the ones where N' is a small constant, or again m is small and the total weight of the elements of $\Phi_{N'}$ is small.

If $\Phi = \Phi_{N'}$ and N' is arbitrary, in the worst case this algorithm becomes less efficient than the Berlekamp-Massey algorithm. The number of operations would be at least $m^2 N$ with $m = |\Phi|$. For the particular case when $N = p^w N'$ with $N' = 2^n - 1$ a (Mersenne) prime and w a small constant, we have $m = \frac{N'-1}{n} + 1$ (the factors of $x^{2^n-1} - 1$ are in this case $x - 1$ and the $\varphi(2^n - 1)/n$ primitive polynomials of degree n as in [44, Theorem 3.5]). Since in this case $m \approx \frac{N}{\log N}$, the complexity of the algorithm becomes in the worst case $\Omega(N^3 / \log^2 N)$ i.e. more than quadratic.

Chapter 6

m-Sequence Stability

Linear feedback shift registers (LFSRs) are frequently used in stream ciphers (see, for example [33] [13] [21] [24] [73]) due to their well understood properties and simplicity of construction in hardware. m-Sequences were discussed by Golomb [29], and have many interesting and well studied properties.

However, to our knowledge, one property that has not been studied is the k -error linear complexity of such sequences. The stability of the linear complexity (i.e. high k -error linear complexity for small values of k) is an important criterion [22] in the design of stream ciphers because if a sequence has a low k -error linear complexity, an attacker could potentially recover easily all but k terms of the sequence.

The k -error linear complexity and the error linear complexity spectrum are very difficult to determine for a general sequence but for some classes of sequences polynomial time algorithms have been found [69] [51] [43] [39]. However, these classes of sequences are usually chosen so that the k -error linear complexity is easy to determine, rather than being chosen because they are used in cryptographic primitives. We begin to rectify that in this chapter by analyzing the k -error complexity of m-sequences, specifically by finding lower bounds on the minimum number of errors that are required to reduce the linear complexity of the sequence.

By analogy to the k -error linear complexity, we can define the k -error period of a sequence, i.e. the minimal period that we can obtain for a sequence by changing up to k terms in each period. As the period of m-sequences is maximal among all sequences of a given linear complexity, m-sequences are often used as components of stream ciphers in order to ensure a large period, see [35]. Therefore it is perhaps even more important in such situations to guarantee the stability of the period rather than of the linear complexity. Moreover for m-sequences the linear complexity cannot be reduced without also reducing the period and therefore the minimum number of errors needed for reducing the period is a lower bound for the minimum number of errors needed for reducing the linear complexity

(Proposition 6.1).

The case of m -sequences with prime period (Section 6.1.1) is relatively easy and we obtain a closed form expression for the k -error linear complexity and the k -error period. When the period is composite, the problem is related to the problem of determining the weight enumerator of minimal cyclic codes (also known as irreducible cyclic codes) (Section 6.1.2). This is a well studied and as yet not fully solved problem (see for example [23]). The weight enumerator is known for certain particular cases, but for the general case it seems no closed form or algorithm better than brute force is known.

If we denote by p a factor of the period of the m -sequence, then we have studied the two extreme cases with regards to the order of 2 modulo p , namely where the order is minimal or maximal. The first of these is where p is a Mersenne number (Section 6.1.3), and the second is where the order of 2 modulo p is $p - 1$ (Section 6.1.4) and in both cases we give exact formulae for the minimum number of errors needed to reduce the period of the m -sequence by p . Other results from coding theory can be similarly transformed to solve more cases, and this is a topic of Chapter 7. In Section 6.1.5 we conjecture that the minimum number of errors needed to reduce the period of an m -sequence is achieved when we reduce the period as little as possible, i.e. we divide the period by its smallest factor. We give several theoretical and experimental results in support of our conjecture. If the conjecture is true then for at least 76% of m -sequences, at least one quarter of the terms in each period must be changed to reduce the period, which implies that these sequences are highly secure from a period (or linear complexity) stability viewpoint.

Finally, in Section 6.2 we study how these results relate to the LFSR components of several cipher systems: the eStream candidates Grain and DECIM^{v2} and also LILI-128 and SSC2. We compute the number of errors needed for reducing their period by several small prime factors. If our conjecture holds, it follows that all of these LFSRs are secure from the point of view of the stability of the period and linear complexity.¹

6.1 k -Error Linear Complexity and Period for Various Classes of m -Sequence

In this chapter we study binary m -sequences. We aim to determine the number of errors needed for reducing the linear complexity of such sequences and the number of errors needed for reducing the period. Due to the fact that m -sequences achieve

¹The main results of this Chapter were originally published in [20].

minimum linear complexity amongst all sequences of that minimum period, in order to reduce the linear complexity of an m-sequence we have to reduce its period as well:

Proposition 6.1. *For any given m-sequence s , the period stability threshold is a lower bound on the linear complexity stability threshold. Moreover if s' is any linearly recurrent sequence with $\text{LC}(s') < \text{LC}(s)$ then $P(s') < P(s)$.*

Proof. It suffices to show the second result. Assume s' is a sequence with $\text{LC}(s') < \text{LC}(s)$, then $P(s') \leq 2^{\text{LC}(s')} - 1 < 2^{\text{LC}(s)} - 1 = P(s)$. \square

6.1.1 Prime period

We will first deal with the relatively easy case when the period of the m-sequence is prime. In this case, we are able to determine not just the period stability threshold, but the full error linear complexity spectrum, which trivially gives the linear complexity stability threshold.

Theorem 6.2. *Consider an m-sequence, s , with $P(s) = m = 2^n - 1$. If m is a Mersenne prime, then the critical points of the k -error linear complexity spectrum of s are: $(0, n)$, $((m - 1)/2, 1)$, $((m + 1)/2, 0)$.*

Proof. From Lemma 2.27 we know that the spectrum will contain at least the three critical points listed in the statement. By Proposition 6.1, the only way to reduce the linear complexity of s is by reducing its period to a factor of m , i.e. to 1, as m is prime. That means s can only become a sequence of all ones (requiring $(m - 1)/2$ changes) or a sequence of all zeros (requiring $(m + 1)/2$ changes). \square

Note that Theorem 6.2 implies that $E_{P(s)} = (m - 1)/2$ and so the linear complexity stability threshold and the period stability threshold are almost half the period length, which implies that such sequences are very stable, in fact, by Lemma 2.27 they are as stable as possible. Since it is possible to construct an m-sequence of period equal to any Mersenne number, the frequency of such sequences among all m-sequences is dependent on the frequency of Mersenne primes among Mersenne numbers. There are no known results about this, but the widely believed Lenstra-Pomerance-Wagstaff Conjecture [62] implies that the proportion of Mersenne primes less than x as a proportion of all Mersenne numbers is $\log \log x / \log x$. This implies that the frequency of these sequences is low, and decreases as we consider longer sequences. Out of the smallest 200 lengths for m-sequences, 13 of them are prime, a proportion of 0.07.

6.1.2 Reducing the period of an m-sequence by an arbitrary factor

For treating the case when the period is composite, reducing the period to a factor r of the original period can be visualized by writing the sequence row-wise in a table of r columns and aiming to make each column of the table contain one single value. We formalize this as follows:

Definition 6.3. For a periodic sequence s , with $P(s) = m$ and $m = qr$ for some integers q, r , we define the r -decimation matrix of s to be the q by r matrix T with entries: $T_{i,j} = s_{ir+j}$ for $i = 0, \dots, q-1$ and $j = 0, \dots, r-1$. That is, we construct T by sequentially filling its rows with the values of s . It will often be useful for us to refer to the columns of T as sequences themselves.

Note that using the notation above, the columns of T are r -regular, improper decimations of s .

Lemma 6.4. The minimum number of errors needed for reducing the period $m = qr$ of a binary sequence s from m to r equals: $E_{P(s)}(r) = \sum_{i=0}^{r-1} \min\{\text{wt}(T_i), q - \text{wt}(T_i)\}$ where T_i is the i -th column of the r -decimation matrix of s .

Proof. The period of the sequence has been reduced to r iff each column of the r -decimation matrix contains only one value. The number of errors needed to make the column T_i contain only zeros is $\text{wt}(T_i)$ and to contain only ones is $q - \text{wt}(T_i)$. \square

Note that an algorithm for computing by brute force the weight of the columns of the decimation matrix is linear in the period length of the sequence. However, for m-sequences the period length is exponentially higher than the degree of the characteristic polynomial, so a more efficient algorithm, or a closed formula, would be preferable.

Theorem 6.5. Let s be a sequence with $P(s) = m$, and $m_1 m_2 | m$.

Then $E_{P(s)}(m/m_1 m_2) \geq E_{P(s)}(m/m_i)$ for $i = 1, 2$.

Proof. Note that a sequence of period $m/m_i m_j$ also has period m/m_i . Therefore $E_{P(s)}(m/m_i m_j)$ errors can change s into a sequence of period m/m_i , and so the result follows. \square

Corollary 6.6. Let s be a sequence with $P(s) = m = p_1^{e_1} p_2^{e_2} \dots p_v^{e_v}$, with p_i prime for all i . Then $E_{P(s)} = \min_i \{E_{P(s)}(m/p_i)\}$.

Corollary 6.6 implies that to determine the period stability threshold for an m-sequence, we will only need to consider reducing the period by a prime factor.

We recall the following results which shed light on the structure of the decimation matrix:

Lemma 6.7. [70] *Assume that T is the r -decimation matrix for an m -sequence s with $P(s) = qr$. Then the columns of T are all generated by a single, irreducible polynomial.*

Lemma 6.8. [70] *Let s be an m -sequence of period m , and assume $m = qr$. If $q = 2^n - 1$ for some n (that is, q is a Mersenne number) then each column of the r -decimation matrix of s will either be an m -sequence or the all zero sequence. Further, each of the m -sequences will be identical, up to a cyclic shift.*

The following is a closely related result from coding theory:

Theorem 6.9. ([45, Theorem 11, Ch. 8, §4]) *Let q, n be integers such that $n = \text{ord}_q(2)$. Let s be an m -sequence with period length $2^n - 1$. The columns of the $(2^n - 1)/q$ -decimation matrix of s are a set of representatives (with respect to the equivalence relation of cyclic shifting) of the non-zero codewords in an irreducible $[q, n]$ cyclic code.*

Please note that the original version of the above theorem uses a matrix which is the transpose of our decimation matrix.

Recall that a cyclic code of length q with generator polynomial $g|x^q - 1$ can be equivalently viewed as the set of sequences of period q with characteristic polynomial equal to the reciprocal of the parity check polynomial $(x^q - 1)/g$. We can define an equivalence relation on the set of sequences generated by a fixed polynomial: two sequences of period t are equivalent if when represented as a finite sequence of length t one can be obtained from the other by a cyclic shift. We can therefore formulate Theorem 6.9 above equivalently as follows:

Corollary 6.10. *Let q, n be integers such that $n = \text{ord}_q(2)$. Let s be an m -sequence with period length $2^n - 1$. The columns of the $(2^n - 1)/q$ -decimation matrix for s are a set of representatives for the set of non-zero sequences generated by a fixed irreducible polynomial of degree n and order q .*

We extend the results above to the case when n is not the order of 2 mod q :

Theorem 6.11. *Let q, n be integers such that $q|2^n - 1$. Let $n' = \text{ord}_q(2)$. Let s be an m -sequence with period length $2^n - 1$. In the $(2^n - 1)/q$ -decimation matrix of s there are $\frac{2^{n-n'} - 1}{q}$ all-zero columns and $2^{n-n'}$ columns from each of the $\frac{2^{n'} - 1}{q}$ equivalence classes of the non-zero sequences generated by a fixed irreducible polynomial of degree n' and order q .*

Proof. We decimate the sequence s in two stages. Let B be the $(2^n - 1)/(2^{n'} - 1)$ decimation matrix of s . Each column of B has length $2^{n'} - 1$ and so by Lemma 6.8 it is either the all-zero column or one fixed m -sequence (possibly shifted). As

in [70, Section IV B] we can count how many of each we have and show that there are $\frac{2^{n-n'}-1}{2^{n'-1}}$ all-zero columns and $2^{n-n'}$ m-sequences. To obtain the $(2^n - 1)/q$ -decimation matrix T of s we can think of concatenating the first $(2^{n'} - 1)/q$ rows of B to obtain the first row of T , then concatenating the next $(2^{n'} - 1)/q$ rows of B to obtain the second row of T and so on. Looking at a particular column of B , say column j , we see that its elements end up as columns $j, j + (2^n - 1)/(2^{n'} - 1), j + 2(2^n - 1)/(2^{n'} - 1), \dots$ of T . Moreover, these columns of T are exactly a $(2^{n'} - 1)/q$ -decimation matrix for the sequence in column j of B . If this sequence is an m-sequence, then by Theorem 6.9 the resulting columns of T are exactly a set of representatives for the equivalence classes of the non-zero sequences generated by a fixed irreducible polynomial. If column j of B is all-zero, then obviously the corresponding columns in T are also all-zero. \square

We have therefore:

Corollary 6.12. *Let s be an m-sequence of length $2^n - 1$ and let q be a factor of $2^n - 1$. Let $n' = \text{ord}_q(2)$. Then:*

$$E_{P(s)}((2^n - 1)/q) = 2^{n-n'} E_{P(s)}((2^{n'} - 1)/q).$$

Recall that the weight enumerator (or weight distribution) of a code C of length m can be defined as the list of integers A_0, A_1, \dots, A_m with A_i equal to the number of codewords in C that have Hamming weight equal to i . In a minimal $[q, n]$ cyclic code each of the q cyclic shifts of a non-zero codeword are distinct, i.e. there are exactly q codewords in each equivalence class, all of the same weight. Therefore, as a consequence of Theorem 6.9 and Corollary 6.12 we have:

Corollary 6.13. *Let s be an m-sequence of length $2^n - 1$ and let q be a factor of $2^n - 1$. Let $n' = \text{ord}_q(2)$. If A_0, A_1, \dots, A_m is the weight enumerator of a minimal $[q, n']$ cyclic code, then*

$$E_{P(s)}((2^n - 1)/q) = 2^{n-n'} \left(\sum_{i=1}^{(q-1)/2} \frac{iA_i}{q} + \sum_{i=(q+1)/2}^q \frac{(q-i)A_i}{q} \right)$$

The problem of finding the weight enumerator for a general cyclic code is a well studied, and yet unsolved, problem in coding theory (see, for example, [23]). There are a number of particular cases for which the problem has been solved, and we examine some of them in the next sections; others can be similarly transferred. However the general case seems difficult as no better solution than brute force (i.e. determining the weight of each column in the decimation matrix) is known.

In view of the corollary above, we suspect that determining $E_{P(s)}(c)$ for s an m-sequence is an equally difficult problem.

In the rest of this section, we will explain how to analyze certain classes of m-sequence. However, in the cases where these results are not applicable, Corollaries 6.12 and 6.13 will allow us to analyze large m-sequences, by transforming the problems into problems about shorter sequences, which may be suitable for a brute force approach. An example of this is given in Section 6.2.

6.1.3 Reducing the period by a Mersenne number

When the period length of an m-sequence is a Mersenne number $2^n - 1$ with n not prime, certain factors of the period are Mersenne numbers themselves and can easily be obtained by factorizing the exponent n . Namely, if n' is a factor of n , then $2^{n'} - 1$ is a factor of $2^n - 1$. In this section we will compute the number of errors needed to reduce the period of an m-sequence by a Mersenne number.

Theorem 6.14. *Consider an m-sequence s with $P(s) = m$ and assume m has a factor of the form $2^{n'} - 1$. Then*

$$E_{P(s)}(m/(2^{n'} - 1)) = (m + 1) \frac{2^{n'-1} - 1}{2^{n'}} = \text{wt}(s) \left(1 - \frac{1}{2^{n'-1}} \right)$$

Proof. Let T_i be the i -th column of the q -decimation matrix T of s . By Lemma 6.8 each T_i is either an m-sequence (and therefore $\text{wt}(T_i) = 2^{n'-1}$) or the all zero sequence (and so $\text{wt}(T_i) = 0$). As in the proof of Theorem 6.14 we can count how many of each we have: $(m + 1)/2^{n'}$ columns of T are m-sequences and the rest are all-zero sequences. Applying Lemma 6.4, $E_{P(s)}(m/(2^{n'} - 1)) = ((m + 1)/2^{n'})(2^{n'-1} - 1)$. \square

Note that Theorem 6.14 can also be viewed as a particular case of Corollary 6.12 for $q = 2^{n'} - 1$, as in that case $E_{P(s)}((2^{n'} - 1)/q) = E_{P(s)}(1) = 2^{n'-1} - 1$ (to reduce the period of an m-sequence to 1 we need to change all zeros into ones).

Example 6.15. *The most important cases will be when the period length m is reduced by a small factor. Theorem 6.14 shows that (for sequences whose period length is divisible by the appropriate factor) $E_{P(s)}(m/3) = (1/4)(m + 1)$, $E_{P(s)}(m/7) = (3/8)(m + 1)$, $E_{P(s)}(m/15) = (7/16)(m + 1)$ and $E_{P(s)}(m/31) = (15/32)(m + 1)$.*

Corollary 6.16. *If the period of an m-sequence s is being reduced by a factor that is a Mersenne prime q , the smallest number of errors required will be when $q = 3$, in which case $(P(s) + 1)/4$ errors are required, i.e. half of the weight of s . As q increases, the number of errors approaches the weight of the sequence.*

Example 6.17. Let s be an m -sequence of period to 4095. Then $E_{P(s)}(4095/3) = E_{P(s)}(1365) = 1024 = (1/2) \text{wt}(s)$ and $E_{P(s)}(4095/7) = E_{P(s)}(585) = 1536 = (3/4) \text{wt}(s)$.

6.1.4 Reducing the period by a prime p with

$$\text{ord}_p(2) = p - 1$$

We will now discuss primes p with the property that $\text{ord}_p(2) = p - 1$, which are those primes where 2 is a primitive root modulo p . For such primes, the factorization of $x^p - 1$ into irreducible factors is the trivial factorization $x^p - 1 = (x - 1)(x^{p-1} + x^{p-2} + \dots + 1)$, i.e. $x^{p-1} + x^{p-2} + \dots + 1$ is an irreducible polynomial of degree $p - 1$. Therefore, if s is an m -sequence of minimal period $2^{p-1} - 1$, by Corollary 6.10 the columns of its $(2^{p-1} - 1)/p$ -decimation matrix are all the non-zero sequences generated by this irreducible polynomial, i.e. sequences obtained by a parity-check bit type equation, so the sum of the bits equals 0. For each even weight $2i$ with $i \neq 0$ there are $\binom{p}{2i}$ such sequences, i.e. $\frac{1}{p} \binom{p}{2i}$ inequivalent sequences (taking equivalence under cyclic shifts). Hence by Lemma 6.4 we have

$$E_{P(s)}((2^{p-1} - 1)/p) = \sum_{i=1}^{\frac{p-1}{2}} \frac{1}{p} \binom{p}{2i} \min(2i, p - 2i).$$

With some combinatorial manipulation we will obtain:

Theorem 6.18. Let p be a prime such that $\text{ord}_p(2) = p - 1$. Let s be an m -sequence of minimal period $2^{p-1} - 1$. Then:

$$E_{P(s)}((2^{p-1} - 1)/p) = 2^{p-2} - \frac{1}{2} \binom{p-1}{\frac{p-1}{2}}.$$

Proof.

$$\begin{aligned}
E_{P(s)}((2^{p-1} - 1)/p) &= \\
&= \sum_{i=1}^{\frac{p-1}{2}} \frac{1}{p} \binom{p}{2i} \min(2i, p - 2i) \\
&= \frac{1}{p} \left(\sum_{i=1}^{\lfloor \frac{p-1}{4} \rfloor} 2i \binom{p}{2i} + \sum_{i=\lfloor \frac{p-1}{4} \rfloor + 1}^{\frac{p-1}{2}} (p - 2i) \binom{p}{2i} \right) \\
&= \frac{1}{p} \left(\sum_{i=1}^{\lfloor \frac{p-1}{4} \rfloor} 2i \binom{p}{2i} + \sum_{i=\lfloor \frac{p-1}{4} \rfloor + 1}^{\frac{p-1}{2}} (p - 2i) \binom{p}{p - 2i} \right) \\
&= \frac{1}{p} \sum_{j=1}^{\frac{p-1}{2}} j \binom{p}{j} \\
&= \sum_{j=1}^{\frac{p-1}{2}} \binom{p-1}{j-1}.
\end{aligned}$$

The last expression is the sum of the combinatorial coefficients in Pascal's triangle on the row $p - 1$ up to but excluding the middle element. If we would also add half of the middle element we would obtain exactly half of the total sum of the row, i.e. 2^{p-2} . Hence the result in the theorem follows. \square

Combining this theorem with Corollary 6.12 we obtain:

Corollary 6.19. *Let s be an m -sequence of minimal period $2^n - 1$ and let p be a prime factor of $2^n - 1$. If $\text{ord}_p(2) = p - 1$ then*

$$\begin{aligned}
E_{P(s)}((2^n - 1)/p) &= 2^{n-1} - 2^{n-p} \binom{p-1}{\frac{p-1}{2}} \\
&= \text{wt}(s) \left(1 - \frac{1}{2^{p-1}} \binom{p-1}{\frac{p-1}{2}} \right)
\end{aligned}$$

By using Stirling's approximation, we can transform this result to obtain $EP_{(2^n-1)/p}(s) \approx \text{wt}(s) \left(1 - \frac{1}{\sqrt{\pi(p-1)}} \right)$ where the accuracy of the approximation depends only on the value of p (and becomes a closer approximation for higher p). This shows that for any fixed n the number of errors will tend to $\text{wt}(s)$ as p becomes larger, just as we saw for the situation when p was a Mersenne Prime. These results, combined with our experimental results lead us to believe that this holds for all m -sequences, but we have not been able to prove this.

6.1.5 The minimum number of errors needed for reducing the period of an m-sequence

In the previous sections we examined the number of errors needed for reducing the period of an m-sequence to specific factors of the original period. In this section we examine the period stability threshold, i.e. the minimum number of errors needed for reducing the period of an m-sequence at all. By Corollary 6.6 this is the minimum among $E_{P(s)}(m/p)$ for the different prime factors p of the period m . We can determine the minimum among different $E_{P(s)}(m/p)$ for those p which fall in the cases of Theorem 6.14 and Corollary 6.19:

Corollary 6.20. *Let s be an m-sequence with $P(s) = m$, and let $p_1 < p_2$ be factors of m . Moreover assume that p_1 and p_2 are such that both satisfy condition (i) or both satisfy condition (ii) or p_1 satisfies condition (ii) and p_2 satisfies condition (i) below:*

(i) *being a Mersenne number*

(ii) *being a prime p such that $\text{ord}_p(2) = p - 1$.*

Then $E_{P(s)}(m/p_1) < E_{P(s)}(m/p_2)$.

Proof. The first situation is immediate, for the second we use the combinatorial inequality $4\binom{2t}{t} > \binom{2(t+1)}{t+1}$. For the last situation, using a Stirling inequality of the form $\binom{2t}{t} \geq \frac{2^{2t-1}}{\sqrt{t}}$, it suffices to prove that $\frac{1}{\sqrt{2(p_1-1)}} > \frac{2}{p_2+1}$, which can be easily verified. \square

It would be tempting to conjecture that $E_{P(s)}(m/p_1) < E_{P(s)}(m/p_2)$ for any prime factors $p_1 < p_2$ of m . However, this is not true, as the following example shows:

Example 6.21. *Consider an m-sequence with period $m = 2^{180} - 1$, $p_1 = 31$ (which falls into case (i) in Corollary 6.20) and $p_2 = 37$ (which falls into case (ii) in Corollary 6.20). We compute $E_{P(s)}(m/31) = 2^{180}(15/32) \approx 7.2 * 10^{53}$ using Theorem 6.14 and $E_{P(s)}(m/37) = 2^{179} - 2^{143} * \binom{36}{18} \approx 5.7 * 10^{53}$ using Corollary 6.19. Hence $E_{P(s)}(m/p_1) > E_{P(s)}(m/p_2)$ holds.*

*However, for our purposes we are only interested in finding the minimum $E_{P(s)}(m/p)$ and in this example neither of these primes achieves it, as $E_{P(s)}(m/3) = 2^{178} \approx 3.8 * 10^{53}$ is lower than both $E_{P(s)}(m/p_1)$ and $E_{P(s)}(m/p_2)$.*

Proposition 6.22. *Let s be an m-sequence with $P(s) = 2^n - 1 = m$, and let p be a prime such that p divides m and $\text{ord}_p(2) = p - 1$. Then the smallest prime factor of m is 3 and $E_{P(s)}(m/3) < E_{P(s)}(m/p)$.*

Proof. The prime factors of m are exactly those primes q with $\text{ord}_q(2)|n$. For the particular p in the statement we have therefore $(p-1)|n$. Since p is odd (obviously 2 is never a factor of $2^n - 1$) that means n is even. On the other hand, $\text{ord}_3(2) = 2$, so 3 must be a factor (the smallest one) of $2^n - 1$ whenever n is even.

Using Corollary 6.19 the inequality becomes $\binom{p-1}{\frac{p-1}{2}} < 2^{p-2}$, which can be easily proved. \square

The results of this section together with an exhaustive search computation for all sequences up to length $2^{17} - 1$ (see Appendix) led us to the following conjecture:

Conjecture 6.23. *Let s be an m -sequence with $P(s) = m$, and p_1 be the smallest prime factor of m . Then if p_2 is any prime prime factor of m , $E_{P(s)}(m/p_1) \leq E_{P(s)}(m/p_2)$.*

Corollary 6.6 becomes:

Corollary 6.24. *Let s be an m -sequence of period m and let p be the smallest prime factor of m . If Conjecture 6.23 holds then $E_{P(s)} = E_{P(s)}(m/p)$.*

Using Theorem 6.14 we obtain therefore:

Corollary 6.25. *Let s be an m -sequence of period m . If the smallest factor of m is a Mersenne prime $2^n - 1$ and Conjecture 6.23 holds then the minimum number of errors needed to reduce the period of s is $E_{P(s)} = (m + 1) \frac{2^{n-1} - 1}{2^n} \geq (1/2) \text{wt}(s)$.*

We will estimate now what proportion of m -sequences are covered by Corollary 6.25, i.e. the proportion of Mersenne numbers that admit a Mersenne prime as their smallest factor. As previously stated, for a given prime p , the Mersenne numbers that are multiples of p are exactly those of the form $2^v - 1$ with $\text{ord}_p(2)|v$. Table 6.1 contains $\text{ord}_p(2)$ for small values of p . Note that $\text{ord}_{2^n-1}(2) = n$ when $2^n - 1$ is prime.

We can compute how many Mersenne numbers have a particular prime as their smallest factor, by using the inclusion-exclusion principle. Specifically, the proportion of Mersenne numbers with some prime p as their smallest factor will be $1/\text{ord}_p(2)$ -th of all those which are not divisible by any prime smaller than p .

The first few values are contained in Table 6.2, and adding them up allows us to say that the proportion of m -sequences whose period length has a Mersenne prime as its smallest factor is at least 0.76. For each of these sequences, the period stability threshold is at least $(P(s) + 1)/4$ (achieved for smallest factor 3). Note that this is a large proportion of errors, as usually the largest number of errors considered is $P(s)/20$ or possibly $P(s)/10$. Out of the smallest 200 lengths of m -sequences, 146 of them have a Mersenne prime as their smallest factor, a proportion

Prime p	$\text{ord}_p(2)$
3	2
5	4
7	3
11	10
13	12
17	8
19	18
23	11
29	28
31	5
37	36
41	20
43	14

Table 6.1: $\text{ord}_p(2)$ for small prime p

Prime p	Proportion of Mersenne numbers having p as smallest factor
3	1/2
7	1/6
31	2/33
127	16/483

Table 6.2: Proportion of Mersenne numbers with certain factors

of 0.73. From a cryptographic standpoint this implies that using m-sequences as primitives in a cipher scheme to ensure a minimum period of the output is a very secure method by this measure, since an unreasonably large number of the bits need to be changed to reduce the period or the linear complexity at all.

We can also determine the proportion of sequences which have a period length that is composite, but that do not have a Mersenne Prime as their smallest factor. Since the proportion of sequences that have prime period length will become arbitrarily small as the lengths considered increase, and we have seen that at least 0.76 of all m-sequences have period length that is divisible by a Mersenne Prime, the proportion that do not cannot be more than 0.24. Out of the smallest 200 lengths for m-sequences, 41 were composite with their smallest factor not a Mersenne prime, a proportion of 0.21.

Example 6.26. *The smallest m-sequence period length which does not have a Mersenne prime as its smallest factor is $2047 = 2^{11} - 1$, with a smallest factor of 23. We have calculated by brute force that the period stability threshold is 869, (when the period is reduced by a factor of 23) which is a large proportion of the weight of the sequence, which is 1024. The next smallest example will occur for the m-sequence of length $8388607 = 2^{23} - 1$, with a smallest factor of 47.*

6.2 Application to Grain and other Stream Ciphers

We will first apply our results to one of the eStream Candidates, namely Grain [33]. Grain is composed of a linear feedback shift register and a non-linear feedback shift register, whose outputs are combined using a non-linear function. We will be looking at the LFSR, which has 80 registers, and a primitive minimal polynomial $f(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80}$. Therefore it generates an m-sequence of period length $2^{80} - 1 (\approx 1.2 * 10^{24})$ which we will refer to as s . It would clearly require a very large amount of processing power to compute by brute force either the k -error linear complexity or the k -error period for s , even for small values of k , but we can use the results of this chapter to study the security of this sequence. To reduce the period of the sequence, we need to know the factors of its period length:

$$2^{80} - 1 = 3 * 5^2 * 11 * 17 * 31 * 41 * 257 * 61681 \\ * 4278255361$$

By using Theorem 6.14, Corollaries 6.19 and 6.12 and a small amount of brute force we can determine the number of errors required to reduce the period by several factors, including all prime factors except 41 and the two largest ones, see Table 6.3. We note that out of the values we computed, the smallest number of errors are needed for reducing the period by 3, the smallest non-trivial factor of the period, as predicted by Conjecture 6.23. It seems unlikely (but we are unable to prove) that any of the remaining primes (41, 61681 or 4278255361) would need less errors. Hence if Conjecture 6.23 holds in this case, the period stability threshold for s is $2^{80}/4 = 2^{78} \approx 3.0 * 10^{23}$ and by Proposition 6.1 this is also a lower bound on the linear complexity stability threshold. Note that since 3 is the smallest possible factor of a Mersenne number, the period stability threshold is as low as it could be for an m-sequence (again assuming Conjecture 6.23 holds). However, since this is half the weight of s , the sequence is still very secure from this point of view. We also note that the LFSR primitive is included as part of Grain not to provide linear complexity for the output, but to ensure that the output of the cipher has a very high minimum period. Therefore, while we have not calculated the linear complexity stability threshold of s in this case it is more important to calculate the period stability threshold which we have done, subject to Conjecture 6.23 holding for this example.

We will also briefly provide some results that can be obtained by applying the results in this paper to other ciphers. Firstly, SSC2 [73] uses an LFSR to

q	$E_{P(s)}((2^{80} - 1)/q)$		
	Formula	Exact	Approx.
3	$2^{80} * (1/4)$	2^{78}	$3.0 * 10^{23}$
5	$2^{80-1} - 2^{80-5} \binom{4}{2}$	$5 * 2^{76}$	$3.8 * 10^{23}$
11	$2^{80-1} - 2^{80-11} \binom{10}{5}$	$193 * 2^{71}$	$4.6 * 10^{23}$
15	$2^{80} * (7/16)$	$7 * 2^{76}$	$5.3 * 10^{23}$
17	$2^{80-8} E_{P(s)}((2^8 - 1)/17)$	$51 * 2^{73}$	$4.8 * 10^{23}$
25	Not Computed		
31	$2^{80} * (15/32)$	$15 * 2^{75}$	$5.7 * 10^{23}$
33	$2^{80-10} * E_{P(s)}((2^{10} - 1)/33)$	$109 * 2^{72}$	$5.1 * 10^{23}$
41	Not Computed		
51	$2^{80-8} * E_{P(s)}((2^8 - 1)/51)$	$115 * 2^{72}$	$5.4 * 10^{23}$
55	Not Computed		
75	Not Computed		
85	$2^{80-8} * E_{P(s)}((2^8 - 1)/85)$	$117 * 2^{72}$	$5.5 * 10^{23}$
93	$2^{80-10} * E_{P(s)}((2^{10} - 1)/93)$	$241 * 2^{71}$	$5.7 * 10^{23}$
123	Not Computed		
155	Not Computed		
165	Not Computed		
187	Not Computed		
205	Not Computed		
255	$2^{80} * (127/256)$	$127 * 2^{72}$	$6.0 * 10^{23}$
257	$2^{80-16} * E_{P(s)}((2^{16} - 1)/257)$	$31029 * 2^{64}$	$5.7 * 10^{23}$

Table 6.3: Errors Required for Reducing the Period of the Grain LFSR

generate an m-sequence of length $2^{127} - 1$, which is prime. Therefore, by Theorem 6.2 the complete error linear complexity spectrum for this sequence is: $(0, 127)$, $(2^{126} - 1, 1)$, $(2^{126}, 0)$ and the period stability threshold is $2^{126} - 1$, i.e. as high as possible.

DECIM^{v2} [13] uses an LFSR to generate an m-sequence of length

$$\begin{aligned} 2^{192} - 1 &= 3^2 * 5 * 7 * 13 * 17 * 97 * 193 * 241 * 257 \\ &\quad * 641 * 673 * 65537 * 6700417 * 22253377 \\ &\quad * 18446744069414584321 \\ &\approx 6.3 * 10^{57} \end{aligned}$$

Again by using Theorem 6.14, Corollaries 6.19 and 6.12 and a small amount of brute force we can compute the results given in Table 6.4.

If Conjecture 6.23 holds for this case, the period stability threshold for s is 2^{190} . Note again that since 3 is a factor of the period, this value is as low as it could be, but still high enough not to represent a threat.

LILI-128 [21] uses two LFSRs both of which generate m-sequences. The first is of length

$$2^{39} - 1 = 7 * 79 * 8191 * 121369 \approx 5.5 * 10^{11}$$

and so by Theorem 6.14 we can say that

$$E_{P(s)}((2^{39} - 1)/7) = 2^{39} * (3/8) \approx 2.1 * 10^{11}$$

which also equals the period stability threshold if Conjecture 6.23 holds for this case. The second is of length $2^{89} - 1$, which is prime, and so by Theorem 6.2 its error linear complexity spectrum is $(0, 89)$, $(2^{88} - 1, 1)$, $(2^{88}, 0)$ and the period stability threshold is $2^{88} - 1$, i.e. as high as possible.

To summarize, all the ciphers considered are highly secure (subject to our conjecture) from the point of view of the period stability of their LFSR component, with SSC2 and LILI-128 even more secure than Grain and DECIM^{v2}.

q	$E_{P(s)}((2^{192} - 1)/q)$		
	Formula	Exact	Approx.
3	$2^{192} * (1/4)$	2^{190}	$1.7 * 10^{57}$
5	$2^{191} - 2^{187} \binom{4}{2}$	$5 * 2^{188}$	$2.0 * 10^{57}$
7	$2^{192} * (3/8)$	$3 * 2^{189}$	$2.4 * 10^{57}$
9	$2^{192-6} * E_{P(s)}((2^6 - 1)/9)$	$23 * 2^{186}$	$2.3 * 10^{57}$
13	$2^{191} - 2^{179} \binom{12}{6}$	$793 * 2^{181}$	$2.4 * 10^{57}$
15	$2^{192} * (7/16)$	$7 * 2^{188}$	$2.7 * 10^{57}$
17	$2^{192-8} * E_{P(s)}((2^8 - 1)/17)$	$51 * 2^{185}$	$2.5 * 10^{57}$
21	$2^{192-6} * E_{P(s)}((2^6 - 1)/21)$	$13 * 2^{187}$	$2.6 * 10^{57}$
35	$2^{192-12} * E_{P(s)}((2^{12} - 1)/35)$	$437 * 2^{182}$	$2.7 * 10^{57}$
39	$2^{192-12} * E_{P(s)}((2^{12} - 1)/39)$	$899 * 2^{181}$	$2.8 * 10^{57}$
45	$2^{192-12} * E_{P(s)}((2^{12} - 1)/45)$	$1823 * 2^{180}$	$2.8 * 10^{57}$
51	$2^{192-8} * E_{P(s)}((2^8 - 1)/51)$	$115 * 2^{184}$	$2.8 * 10^{57}$
63	$2^{192} * (31/64)$	$31 * 2^{186}$	$3.0 * 10^{57}$
65	$2^{192-12} * E_{P(s)}((2^{12} - 1)/65)$	$917 * 2^{181}$	$2.8 * 10^{57}$
85	$2^{192-8} * E_{P(s)}((2^8 - 1)/85)$	$117 * 2^{184}$	$2.9 * 10^{57}$
91	$2^{192-12} * E_{P(s)}((2^{12} - 1)/91)$	$1869 * 2^{180}$	$2.9 * 10^{57}$
97	Not Computed		
105	$2^{192-12} * E_{P(s)}((2^{12} - 1)/105)$	$235 * 2^{183}$	$2.9 * 10^{57}$
117	$2^{192-12} * E_{P(s)}((2^{12} - 1)/117)$	$1889 * 2^{180}$	$2.9 * 10^{57}$
119	Not Computed		
153	Not Computed		
193	Not Computed		
195	$2^{192-12} * E_{P(s)}((2^{12} - 1)/195)$	$959 * 2^{181}$	$2.9 * 10^{57}$
221	Not Computed		
241	Not Computed		
255	$2^{192} * (127/256)$	$127 * 2^{184}$	$3.1 * 10^{57}$
257	$2^{192-16} * E_{P(s)}((2^{16} - 1)/257)$	$31029 * 2^{176}$	$3.0 * 10^{57}$

Table 6.4: Errors Required for Reducing the Period of the DECIM^{v2} LFSR

Chapter 7

Further results on m-sequence stability

7.1 Period Stability Results

In this chapter we will use the same notation as Chapter 6. First we recall Theorem 6.11 (slightly rephrased):

Theorem. *Given an irreducible cyclic $[q, m]$ code \mathcal{C} , the codewords of \mathcal{C} will correspond with the columns of the q -decimation table of an m -sequence of length $2^m - 1$. Therefore determining the weight enumerator of \mathcal{C} is sufficient to determine $E_{P(s)}((2^m - 1)/q)$. For ease of notation we will define N as $(2^m - 1)/q$.*

In Chapter 6 we used this theorem to show how the weight enumerator of irreducible cyclic codes can be used to determine the number of errors needed to reduce the period of an m -sequence by a given amount. As the weight enumerator is not known for the general case, we must restrict ourselves to specific cases where it is known. Throughout we will use the notation introduced in Definition 2.40

7.1.1 Two Weight Irreducible Cyclic Codes

Various classes of irreducible cyclic codes have been studied, and there has been particular success in determining the weight enumerator polynomial for two weight codes, that is codes where the weight of the codewords takes at most two non-zero values. These codes are described by Schmidt and White [67], and three cases are considered. The first is the Subfield Case, which corresponds to the situation whereby the decimated sequences are also m -sequences. This is the case that was discussed in Chapter 6, and we will not go into any more detail here. The second case studied by Schmidt and White is where N divides $2^j + 1$ for some j that is a divisor of $m/2$ and is known as the Semi-Primitive case. It was

originally studied by (amongst others) Baumert and McEliece [11], and the weight enumerator polynomial has been determined. The final case that is considered by Schmidt and White is that of all two weight codes that do not fall under either of the previous categories, and is denoted as *exceptional*. However, none of the known exceptional cases are binary codes, and it is conjectured that all exceptional cases have been found. Therefore the only case that we need to consider from the paper of Schmidt and White is the semi-primitive case. For ease of notation, however, we will present the results as they were originally given in the paper of Baumert and McEliece. Throughout this section, we will restrict ourselves to the binary case.

Definition 7.1. *Let $N = (2^m - 1)/q$ and assume \mathcal{C} is an irreducible cyclic $[q, m]$ code. When N divides $2^j + 1$ for some j that is a divisor of $m/2$, this is known as the Semi-Primitive case.*

Theorem 7.2. *([11, Theorem 6]) The weight of the codewords, s of an $[q, m]$ semi-primitive irreducible cyclic code over a field of characteristic 2 has the following form:*

Class 1 (containing q codewords):

$$\text{wt}(s) = \frac{r - 1}{2N} + \frac{1 \pm (N - 1)\sqrt{r}}{2N}$$

Class 2 (containing $q(N - 1)$ codewords):

$$\text{wt}(s) = \frac{r - 1}{2N} + \frac{1 \mp \sqrt{r}}{2N}$$

where $N = (2^m - 1)/q$, $r = 2^m$ and the sign of \pm is uniquely determined by the requirement that the weight is an integer.

However, as we are only interested in determining the weight of the columns of the n -decimation matrix, we only need to consider the weights of the cyclic representatives of the code. That is, if two codewords only differ by a cyclic shift, we do not consider them as distinct. This gives us:

Corollary 7.3. *When $N|2^j + 1$ for some $j|(m/2)$ then the sequences generated by an irreducible characteristic polynomial have exactly two different weights. Specifically, there will be one sequence with weight:*

$$\text{wt}(s) = \frac{r - 1}{2N} + \frac{1 \pm (N - 1)\sqrt{r}}{2N}$$

and $N - 1$ sequences with weight:

$$\text{wt}(s) = \frac{r-1}{2N} + \frac{1 \mp \sqrt{r}}{2N}$$

By using the formula given in Lemma 6.4:

$$E_{P(s)}(N) = \sum_{i=0}^{N-1} \min(\text{wt}(S_i), n - \text{wt}(S_i))$$

Corollary 7.3 allows us to determine the number of errors needed to reduce the period of an m -sequence to at most N .

Example 7.4. Consider the m -sequence of length 255. We will reduce its period to 17. In this case, $N = 17$, $q = 15$, $m = 8$ and we can see that $j = 4$ satisfies the condition in Definition 7.1. Therefore, by Corollary 7.3, there will be 1 sequence of weight:

$$\frac{255}{34} + \frac{1 \pm (16)\sqrt{256}}{34}$$

and 16 sequences of weight:

$$\frac{255}{34} + \frac{1 \mp \sqrt{256}}{34}$$

Trying both possibilities for \pm determines that there is 1 sequence of weight 0 and 16 sequences of weight 8. This, in turn, implies that

$$1 * \min(0, 15 - 0) + 16 * \min(8, 15 - 8) = 16 * 7 = 112$$

errors are required to reduce the period, which can be verified by comparison with Appendix A.

7.1.2 Prime factors N , such that $N \equiv 3 \pmod{4}$ and

$$\text{ord}_N(2) = (N - 1)/2$$

Baumert and Mykkeltveit [12] studied the problem of determining the weight enumerator polynomial for irreducible cyclic codes with $N \equiv 3 \pmod{4}$ and $\text{ord}_N(p) = (N - 1)/2$ for some prime p . For our cases we will just restrict our studies to the case when $p = 2$. We will need the following notation:

Notation 7.5. Let $t = t_0 2^0 + t_1 2^1 + t_2 2^2 + \dots$ be the base 2 expansion of t . Then $w_2(t) = t_0 + t_1 + t_2 + \dots$

Specifically Baumert and Mykkeltveit proved the following:

Theorem 7.6. (Theorem from page 131 of [12]) *Let N be a prime such that $N \equiv 3 \pmod{4}$ and $N \neq 3$. Let $k = \text{ord}_N(2) = (N - 1)/2$, $a = w_2(q)$, $r = 2^k$ and c and d be the unique positive odd integers that satisfy $c^2 + Nd^2 = 2^{(k-2a)+2}$. Then there are three distinct weights of non-zero codewords of the associated $[q, k]$ irreducible cyclic code:*

Class 0 (containing n codewords):

$$\text{wt}(s) = \frac{2r \mp 2^a c(N - 1)}{4N}$$

Class 1 (containing $q(N - 1)/2$ codewords):

$$\text{wt}(s) = \frac{2r \mp 2^a (dN - c)}{4N}$$

Class -1 (containing $q(N - 1)/2$ codewords):

$$\text{wt}(s) = \frac{2r \pm 2^a (dN + c)}{4N}$$

Again, as before, we are not interested in cyclic shifts of a codeword, so the number of sequences will be q times less than the corresponding number of codewords. This gives us:

Corollary 7.7. *Using the terminology from Theorem 7.6 sequences generated by an irreducible polynomial which are of length N will have three distinct weights. There will be one sequence of weight:*

$$\frac{2r \mp 2^a c(N - 1)}{4N}$$

$(N - 1)/2$ sequences of weight:

$$\frac{2r \mp 2^a (dN - c)}{4N}$$

and $(N - 1)/2$ sequences of weight:

$$\frac{2r \pm 2^a (dN + c)}{4N}$$

Again, by using the formula given in Lemma 6.4:

$$E_{P(s)}(N) = \sum_{i=0}^{N-1} \min(\text{wt}(S_i), n - \text{wt}(S_i))$$

Corollary 7.7 allows us to determine the number of errors needed to reduce the

period of an m-sequence to a factor N .

Example 7.8. Consider the m-sequence of length 2047, and we will reduce its period to 23. This satisfies the conditions of Theorem 7.6 as $23 \equiv 3 \pmod{4}$ and $\text{ord}_N(2) = 11 = (N - 1)/2$. We can calculate that $a = w_2(89) = 4$, and solving $c^2 + 23d^2 = 2^{11-8+2}$ gives us $c = 3, d = 1$. Therefore, by Corollary 7.7, there will be 1 sequence of weight:

$$\frac{2 * 2048 \mp 2^4 * 3 * 22}{4 * 34}$$

11 sequences of weight:

$$\frac{2 * 2048 \mp 2^4(1 * 23 - 3)}{4 * 34}$$

and 11 sequences of weight:

$$\frac{2 * 2048 \pm 2^4(1 * 23 + 3)}{4 * 34}$$

Trying both possibilities for the \mp and \pm shows us that the weights of the three classes of codewords are 56, 48 and 40 respectively. Therefore

$$E_{P(s)}(23) = 1 * \min(56, 89 - 56) + 11 * \min(48, 89 - 48) + 11 * \min(40, 89 - 40) = 924$$

which again agrees with the results in Appendix A.

Chapter 8

Conclusions

Pseudo-random sequences are used as a crucial part of stream ciphers. As such it is important to be able to determine the properties of such a sequence, and so to determine just how predictable a sequence is, under a number of different models. In this thesis, we have explored a number of different randomness measures, and have found new ways of determining them for certain classes of sequence.

In Chapter 4 we have given details of a new way of attempting to determine the k -error linear complexity of a sequence. We detailed two algorithms, which are based on the same principles, run in $\mathcal{O}(N^4)$ time, and search an exponentially large space of error sequences. We also provided examples of the size of the space searched in the examples where the length of the sequence was 32, 48 or 60, the sequence was taken over $\text{GF}(2)$, where k was not greater than 3, and for a variety of sequence complexities.

In Chapter 5 we proposed algorithms for computing the linear complexity and minimal polynomial for sequences which admit as a characteristic polynomial a power of a fixed irreducible polynomial f . They work for any field of finite characteristic and we do not necessarily need the whole period of the sequence. For $f = x - 1$ our algorithms reduce to the algorithms of Games-Chan [28], Ding *et al.* [22] and Meidl [54]. When we have a whole period of the sequence, we can also apply our algorithms to the case where the characteristic polynomial is a product of powers of a small number of known irreducible polynomials. All our algorithms have linear computational complexity (when assuming the irreducible polynomials are fixed).

In Chapter 6 we have studied the k -error linear complexity and k -error period of m-sequences. We have shown that although the general problem of determining these values is likely to be difficult, there are certain cases where we can find results. We have fully solved the case where the period length of the m-sequence is prime. We have shown how in general the problem of determining the number of errors needed for reducing the period of a sequence s by a factor q can be

reduced to an equivalent problem for shorter m-sequences and we have provided a closed form expression for this number when q is a Mersenne number or a prime with $\text{ord}_q(2) = q - 1$. Subject to a conjecture, for a large proportion ($> 76\%$) of m-sequences we have provided results for the number of errors needed to reduce the period at all. Finally we have applied these results to the LFSRs of several stream ciphers, and seen that, subject to our conjecture, they are all very secure from this particular viewpoint, with DECIM^{v2} and LILI-128 even more so than Grain and SSC2.

8.1 Future Work

There are a number of possible ways that the work in this thesis could be extended in future. Arguably the most interesting of these is to study how the results of Chapters 6 and 7 could be extended to sequences over larger fields. Increasingly in modern cryptographic proposals, the pseudo-random sequences that are used are over fields whose size is a power of a small prime, such as $\text{GF}(2^r)$ or $\text{GF}(3^r)$ for some integer r . I feel it would be relatively straightforward to extend our results to these fields, but unfortunately did not have time to do so during my studies. Alternative ways to extend these results would be to look at sequences over simpler fields, such as $\text{GF}(p)$ for some small prime p . Another possibility would be to look at other sequences for which the auto-correlation has been well studied, such as those discussed by No *et al.* [59], Gong and Xiao [30], Alhakim and Akinwande [5], Boztaş and Parampalli [17] or others as this is the main property which was exploited in determining the results of Chapters 6 and 7.

Other potential ways in which our work could be extended would be to study the way that period, linear complexity, k -error linear complexity or k -error period are inherited when sequences are combined. Although pseudo-random sequences are often combined in non-linear ways with other primitives as part of stream cipher proposals, it was only recently that Hu and Gong [37] showed that in certain models, the period of the sequence is a lower bound on the period of the output of the cipher. As this work only concerned two specific stream ciphers, and only the properties of the period length, there is much room for future investigations into other sequence measures, and other ways of combining cryptographic properties.

We also feel that there is a potential for a lot more research to be done in general on the k -error period of sequences. Whilst previously, the k -error linear complexity of a sequence has been a popular topic for research, we feel that due to the straightforward nature of the period length, the k -error period has been overlooked. However, as it is the period length that most LFSR primitives are relied upon to provide, it seems that this should be the focus of the community in

ensuring that cryptographic schemes really do have the properties that they are claimed to, and that it is not possible to vastly reduce the period of the output by simply tolerating a small number of errors in the decrypt.

References

- [1] A. Alecu. Algorithms for the k -error linear complexity of cryptographic sequences over finite fields. *Loughborough University*, 2008.
- [2] A. Alecu and A. Sălăgean. A genetic algorithm for computing the k -error linear complexity of cryptographic sequences. *IEEE Congress on Evolutionary Computation, 2007*, pp. 3569–3576, 2007.
- [3] A. Alecu and A. Sălăgean. Modified Berlekamp-Massey Algorithm for Approximating the k -Error Linear Complexity of Binary Sequences. 11th IMA International Conference on Cryptography and Coding, 2007, pp. 220–232, 2007.
- [4] A. Alecu and A. Sălăgean. An Approximation Algorithm for Computing the k -Error Linear Complexity of Sequences Using the Discrete Fourier Transform. *IEEE International Symposium on Information Theory (ISIT), 2008*, pp. 2414–2418, 2008.
- [5] A. Alhakim and M. Akinwande. A recursive construction of nonbinary de Bruijn sequences. *Design Codes and Cryptography*, vol. 60, pp. 155–169, 2010.
- [6] H. Aly and A. Winterhof. On the k -error linear complexity over \mathbb{F}_p of Legendre and Sidelnikov sequences. *Design Codes and Cryptography*, vol. 40, pp. 369–374, 2006.
- [7] S. K. Arora and M. Pruthi. Minimal Cyclic Codes of Length $2p^n$. *Finite Fields and Their Applications*, vol. 5, pp. 177–187, 1999.
- [8] Y. Aubry and P. Langevin. On the Weights of Binary Irreducible Cyclic Codes. *Lecture Notes in Computer Science*, vol. 3969, pp. 46–54, 2006.
- [9] S. Babbage and M. Dodd. The stream cipher MICKEY 2.0. *Lecture Notes in Computer Science*, vol. 4986, pp. 191–209, 2008.

- [10] G. K. Bakshi and M. Raka. Minimal cyclic codes of length p^nq . *Finite Fields and Their Applications*, vol. 9, pp. 432–448, 2003.
- [11] L. D. Baumert and R. J. McEliece. Weights of Irreducible Cyclic Codes. *Information and Control*, Vol. 20, pp. 158–175, 1972.
- [12] L. D. Baumert and J. Mykkeltveit. Weight Distribution of Some Irreducible Cyclic Codes. *JPL Technical Report*, Vol. 16, pp. 128–131, 1973.
- [13] C. Berbain, O. Billet, A. Canteaut, N. Courtois, B. Debraize, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Silbert. DECIM^{v2}. *eStream Candidate, ECRYPT Stream Cipher Workshop SKEW 2005*.
- [14] C. Berbain, O. Billet, A. Canteaut, N. Courtois, B. Debraize, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Silbert. SOSEMANUK, a Fast Software-Oriented Stream Cipher. *Lecture Notes in Computer Science*, vol. 4986, pp. 98–118, 2008.
- [15] E. R. Berlekamp. Algebraic Coding Theory. McGraw-Hill 1968.
- [16] S. R. Blackburn. A Generalisation of the Discrete Fourier Transform: Determining the Minimal Polynomial of a Periodic Sequence. *IEEE Transactions on Information Theory*, vol. 40, pp. 1702–1704, 1994.
- [17] S. Boztaş and U. Parampalli. On the Relative Abundance of Nonbinary Sequences with Perfect Autocorrelations. *IEEE International Symposium on Information Theory Proceedings (ISIT) 2011*, pp. 464–468, 2011.
- [18] A. Braeken, J. Lano, N. Mentens, B. Preneel and I. Verbauwhede. SFINKS: A Synchronous Stream Cipher for Restricted Hardware Environments. *eStream Candidate, ECRYPT Stream Cipher Workshop SKEW 2005*.
- [19] A. J. Burrage, A. Sălăgean and R. C.-W. Phan. Linear Complexity for Sequences with Characteristic Polynomial f^v . *IEEE International Symposium on Information Theory Proceedings (ISIT) 2011*, pp. 688–692, 2011.
- [20] A. J. Burrage, A. Sălăgean and R. C.-W. Phan, On the Stability of m -Sequences. *Proceedings of the 13th IMA International Conference on Cryptography and Coding, Lecture Notes in Computer Science*, vol. 7089, pp. 259–274, 2011.
- [21] E. Dawson, A. Clark, J. Golić, W. Millan, L. Penna, and L. Simpson. The LILI-128 Keystream Generator. *Proceedings 1st NESSIE Workshop*, 2000.

- [22] C. Ding, G. Xiao, and W. Shan. The Stability Theory of Stream Ciphers. *Lecture Notes in Computer Science* vol. 561, Springer Verlag, 1991.
- [23] C. Ding. The Weight Distribution of Some Irreducible Cyclic Codes. *IEEE Transactions on Information Theory*, vol. 55, pp. 955–960, 2009.
- [24] P. Ekdahl and T. Johansson. Another Attack on A5/1. *IEEE Transactions on Information Theory*, vol. 49, pp. 284–289, 2003.
- [25] The eStream Project. <http://www.ecrypt.eu.org/stream/project.html> .
- [26] T. Etzion, N. Kalouptsidis, N. Kolokotronis, K. Limniotis and K.G. Paterson. Properties of the Error Linear Complexity Spectrum. *IEEE Transactions on Information Theory*, vol. 55, pp. 4681–4686, 2009.
- [27] H. Fell. Linear complexity of transformed sequences. *EUROCODE '90*, pp. 205–214, 1991.
- [28] R. Games and A. Chan. A fast algorithm for determining the complexity of a binary sequence with period 2^n . *IEEE Transactions on Information Theory*, vol. 29, pp. 144–146, 1983.
- [29] S. W. Golomb. Shift Register Sequences. *Aegean Park Press*, 1982.
- [30] G. Gong and G. Z. Xiao. Synthesis and Uniqueness of m -sequences over $\text{GF}(q^n)$ as n -Phase Sequences over $\text{GF}(q)$. *IEEE Transactions on Information Theory*, vol. 42, pp. 2501–2505, 1994.
- [31] P. Hawkes and G. Rose. Primitive Specification for SOBER-128. *IACR ePrint archive*, 2003.
- [32] Y. K. Han, J. -H. Chung and K. Yang. On the Error Linear Complexity of Periodic Binary Sequences. *IEEE Transactions on Information Theory*, vol. 53, pp. 2297–2304, 2007.
- [33] M. Hell, T. Johansson, and W. Meier. Grain - A Stream Cipher for Constrained Environments. *International Journal of Wireless and Mobile Computing, Special Issue on Security of Computer Network and Mobile Systems*, 2006.
- [34] T. Helleseth, T. Kløve and J. Mykkeltveit. The Weight Distribution of Irreducible Cyclic Codes with Block Lengths $n_1((q^l - 1)/N)$. *Discrete Mathematics*, vol. 18, pp. 179–211, 1977.

- [35] H. Hu. Periods on Two Kinds of Nonlinear Feedback Shift Registers with Time Varying Feedback Functions. *Technical Reports, Center for Applied Cryptographic Research*, 2011.
- [36] H. Hu and D. Feng. On the 2-Adic Complexity and the k -Error 2-Adic Complexity of Periodic Binary Sequences. *IEEE Transactions on Information Theory*, Vol. 54, pp. 874–883, 2008.
- [37] H. Hu and G. Gong. Periods on Two Kinds of Nonlinear Feedback Shift Registers with Time Varying Feedback Functions. *Technical Reports of CACR*, 2011.
- [38] C. J. .A. Jansen, T. Helleseth and A. Kholosha. Cascade Jump Controlled Sequence Generator and Pomaranch Stream Cipher. *Lecture Notes in Computer Science*, vol. 4986, pp. 224–243, 2008.
- [39] T. Kaida, S. Uehara, and K. Imamura. An algorithm for the k -error linear complexity of sequences over $GF(p^m)$ with period p^n , p a prime. *Information and Computation*, vol. 151, pp. 134–147, 1999.
- [40] J.-H. Kim and H.-Y. Song. On the linear Complexity of Hall’s Sextic Residue Sequences. *IEEE Transactions on Information Theory*, Vol. 47, pp. 2094–2096, 2001.
- [41] A. Klapper and M. Goresky. Feedback Shift Registers, 2-Adic Span, and Combiners with Memory *Journal of Cryptography*, vol. 10, pp. 111–147, 2010.
- [42] K. Kurosawa, F. Sato, T. Sakata and W. Kishimoto. A relationship between linear complexity and k -error linear complexity. *IEEE Transactions on Information Theory*, Vol. 46, pp. 694–698, 2000.
- [43] A. G. B. Lauder and K. G. Paterson. Computing the error linear complexity spectrum of a binary sequence of period 2^n . *IEEE Transactions on Information Theory*, vol. 49, pp. 273–280, 2003.
- [44] R. Lidl and H. Niederreiter, Introduction to Finite Fields and Their Applications. *Cambridge University Press*, 1994.
- [45] F. J. MacWilliams and N. J. A. Sloane. The Theory of Error-Correcting Codes. *North-Holland*, 1978.
- [46] J. MacWilliams and J. Seery. The Weight Distributions of Some Minimal Cyclic Codes. *IEEE Transactions on Information Theory*, vol. 27, pp. 796–806, 1981.

- [47] J. L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, vol. 15, pp. 122–127, 1969.
- [48] J. L. Massey. The Discrete Fourier Transform in Coding and Cryptography. *IEEE Information Theory Workshop, (ITW) 1998*, pp 9–11, 1998.
- [49] J. L. Massey and S. Serconek. Linear Complexity of Periodic Sequence: A General Theory. *Advances in Cryptology CRYPTO 96*, vol. 1109, pp. 358–371, 1996.
- [50] R. J. McEliece and H. Rumsey. Euler Products, Cyclotomy and Coding. *Journal of Number Theory*, vol. 4, pp.302–311, 1972.
- [51] W. Meidl, H. Aly, and A. Winterhof. On the k -error linear complexity of cyclotomic sequences. *Journal Mathematical Cryptography*, 2007.
- [52] W. Meidl. On the Stability of 2^n -Periodic Binary Sequences. *IEEE Transactions on Information Theory* vol. 51, pp. 1151–1155, 2005.
- [53] W. Meidl. Reducing the calculation of the linear complexity of $u2^v$ -periodic binary sequences to Games-Chan algorithm. *Designs Codes and Cryptography*, vol. 46, pp. 57–65, 2008.
- [54] W. Meidl. How to determine linear complexity and k -error linear complexity in some classes of linear recurring sequences. *Cryptography and Communications*, vol. 1, pp. 117–133, 2009.
- [55] W. Meidl and H. Niederreiter. On the expected value of the linear complexity and the k -error linear complexity of periodic sequences. *IEEE Transactions on Information Theory* vol. 48, pp. 2817–2825, 2002.
- [56] W. Meidl and H. Niederreiter. Periodic Sequences with Maximal Linear Complexity and Large k -Error Linear Complexity. *Applicable Algebra in Engineering, Communication and Computing*, vol. 14, pp. 273–286, 2003.
- [57] F. Miller. Telegraphic code to insure privacy and secrecy in the transmission of telegrams. *Cipher and Telegraph Codes*, CM Cornwell, 1882.
- [58] H. Niederreiter. Periodic Sequences With Large k -Error Linear Complexity. *IEEE Transactions on Information Theory*, vol. 49, pp. 501–505, 2003.
- [59] J.-S. No, S. W. Golomb, G. Gong, H.-K. Lee and P. Gaal. Binary Pseudorandom Sequences of Period $2^n - 1$ with Ideal Autocorrelation. *IEEE Transactions on Information Theory* vol. 44, pp. 814–817, 1998.

- [60] The On-Line Encyclopedia of Integer Sequences. <http://www.oeis.org>.
- [61] K. G. Paterson. Root Counting, the DFT and the Linear Complexity of Nonlinear Filtering. *Designs, Codes and Cryptography*, vol. 14, pp. 247–259, 1998.
- [62] C. Pomerance. Recent developments in primality testing. *The Mathematical Intelligencer*, vol. 3, pp. 97–105, 1981.
- [63] M. Pruthi and S. K. Arora. Minimal Codes of Prime Power Length. *Finite Fields and Their Applications*, vol. 3, pp. 99–113, 1997.
- [64] A. Sălăgean. On the computation of the linear complexity and the k -error linear complexity of binary sequences with period a power of two. *IEEE Transactions on Information Theory*, vol. 51, pp. 1145–1150, 2005.
- [65] A. Sălăgean. An Algorithm for Computing Minimal Bidirectional Linear Recurrence Relations. *IEEE Transactions on Information Theory*, Vol. 55, pp. 4695–4700, 2009.
- [66] A. Sălăgean and A. Alecu. An Improved Approximation Algorithm for Computing the k -Error Linear Complexity of Sequences Using the Discrete Fourier Transform *Sequences and Their Applications (SETA) 2010*, pp. 151–165, 2010.
- [67] B. Schmidt and C. White. All Two-Weight Irreducible Cyclic Codes. *Finite Fields and Their Applications*, Vol. 8, pp. 1–17, 2000.
- [68] C. E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, vol. 28, 1949.
- [69] M. Stamp and C. F. Martin. An algorithm for the k -error linear complexity of binary sequences of period 2^n . *IEEE Transactions on Information Theory*, vol. 39, pp. 1398–1401, 1993.
- [70] F. Surböck and H. Weinrichter. Interlacing Properties of Shift-Register Sequences with Generator Polynomials Irreducible over $\text{GF}(p)$. *IEEE Transactions on Information Theory*, vol. 24, pp. 386–389, 1978.
- [71] G. Xiao, S. Wei, K. Y. Lam and K. Imamura. A fast algorithm for determining the linear complexity of a sequence with period pn over $\text{GF}(q)$. *IEEE Transactions on Information Theory*, vol. 46, pp. 2203–2206, 2000.
- [72] B. Zhang. New Cryptanalysis of Irregularly Decimated Stream Ciphers. *Lecture Notes in Computer Science*, vol. 5876, pp. 449–465, 2009.

- [73] M. Zhang, C. Carroll, and A. H. Chan. The Software-Oriented Stream Cipher SSC2. *Fast Software Encryption Workshop*, 2000.
- [74] J. Zhou. On the k -error linear complexity of sequences with period $2p^n$ over $\text{GF}(q)$. *Designs Codes and Cryptography*, vol. 58, pp. 279–296, 2010.

Appendix A

k -Error Period Tables

We computed an exhaustive table for the the m-sequences of period up to $m = 2^{17} - 1$. For each factor q of m we computed $E_{P(s)}(m/q)$, i.e. the number of errors needed for reducing the period to m/q . Conjecture 6.23 was verified in all these cases. We also computed the linear complexity $LC(s')$ of the sequence s' obtained by making these errors. Note that this linear complexity is not always lower than the complexity n of the original sequence. This shows that while we know that for reducing the linear complexity it is necessary to reduce the period (Proposition 6.1), this is not always sufficient and further errors may be needed to reduce the linear complexity. In other words, the period stability threshold is a lower bound for the linear complexity stability threshold, but not a tight one. To aid the reader, we also included the section of the paper from which the results were used to obtain the data in each row.

m	n	q	$E_{P(s)}(m/q)$	$LC(s')$	Relevant Section
15	4	3	4	4	6.1.3
		5	5	3	6.1.4
		15	7	1	Lemma 2.27
31	5	31	15	1	6.1.1
63	6	3	16	9	6.1.3
		7	24	8	6.1.3
		9	23	4	Brute Force
		21	26	2	Brute Force
		63	31	1	Lemma 2.27
127	7	127	63	1	6.1.1
255	8	3	64	16	6.1.3
		5	80	24	6.1.4
<i>continued on next page</i>					

m	n	q	$E_{P(s)}(m/q)$	$LC(s')$	Relevant Section
		15	112	16	6.1.3
		17	102	6	Brute Force
		51	115	5	Brute Force
		85	117	3	Brute Force
		255	127	1	Lemma 2.27
511	9	7	192	27	6.1.3
		73	235	4	Brute Force
		511	255	1	Lemma 2.27
1023	10	3	256	25	6.1.3
		11	386	67	6.1.4
		31	480	32	6.1.3
		33	436	15	Brute Force
		93	482	10	Brute Force
		341	490	2	Brute Force
		1023	511	1	Lemma 2.27
2047	11	23	869	45	Brute Force
		89	924	11	Brute Force
		2047	1023	1	Lemma 2.27
4095	12	3	1024	36	6.1.3
		5	1280	99	6.1.4
		7	1536	64	6.1.3
		9	1472	112	Corollary 6.12 & Brute Force
		13	1586	161	6.1.4
		15	1792	81	6.1.3
		21	1664	16	Corollary 6.12 & Brute Force
		35	1748	60	Brute Force
		39	1798	27	Brute Force
		45	1823	64	Brute Force
		63	1984	65	6.1.3
		65	1834	36	Brute Force
		91	1869	19	Brute Force
		105	1880	12	Brute Force
<i>continued on next page</i>					

m	n	q	$E_{P(s)}(m/q)$	$LC(s')$	Relevant Section
		117	1889	19	Brute Force
		195	1918	5	Brute Force
		273	1945	11	Brute Force
		315	1988	12	Brute Force
		455	1991	9	Brute Force
		585	1972	3	Brute Force
		819	1996	4	Brute Force
		1365	2005	3	Brute Force
		4095	2047	1	Lemma 2.27
8191	13	8191	4095	1	6.1.1
16383	14	3	4096	49	6.1.3
		43	7189	192	Brute Force
		127	8064	128	6.1.3
		129	7583	78	Brute Force
		381	8066	42	Brute Force
		5461	8106	2	Brute Force
		16383	8191	1	Lemma 2.27
32767	15	7	12288	125	6.1.3
		31	15360	243	6.1.3
		151	15270	53	Brute Force
		217	15544	30	Brute Force
		1057	15964	10	Brute Force
		4681	16233	7	Brute Force
		32767	16383	1	Lemma 2.27
65536	16	3	16384	64	6.1.3
		5	20480	288	6.1.4
		15	28672	256	6.1.3
		17	26112	96	Corollary 6.12 & Brute Force
		51	29440	320	Corollary 6.12 & Brute Force
		85	29952	288	Corollary 6.12 & Brute Force
		255	32512	256	6.1.3
<i>continued on next page</i>					

m	n	q	$E_{P(s)}(m/q)$	$LC(s')$	Relevant Section
		257	31029	123	Brute Force
		771	31852	44	Brute Force
		1285	32095	9	Brute Force
		3855	32527	17	Brute Force
		4369	32439	5	Brute Force
		13107	32563	4	Brute Force
		21845	32597	3	Brute Force
		65536	32527	1	Lemma 2.27
131071	17	131071	65536	1	6.1.1

Table A.1: m-Sequence Period Stability Results